DEGREE PROJECT IN COMMUNICATION NETWORKS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Hardened Registration Process for Participatory Sensing

Jatesada Borsub

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Jatesada Borsub <jatesada@kth.se>
Master’s Programme, Systems, Control and Robotics
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner

Panos Papadimitratos
Networked Systems Security Group
KTH Royal Institute of Technology

Supervisor

Hongyu Jin
Networked Systems Security Group
KTH Royal Institute of Technology

Acknowledgements

This study was completed towards completing the requirements of the Master's Program in Systems, Control and Robotics at KTH Royal Institute of Technology.

I would like to give special thanks to my examiner, Panos Papadimitratos, and my supervisor, Hongyu Jin, for all their support be it understanding, suggestions or guidance during all the phases of this work from the beginning until the end of the work.

I would also like to thank people in Networked Systems Security Group at KTH who gave me commends and ideas to improve my work.

Last but not least, I would like to thank Tattaton Mettaphun, Boonyakiat Rungsiwothinan, Wichapong Kua-Arun, Surapat Somsri, Nonthawun Junsrisuk, my family and my girlfriend who supported me during my endeavour of studying. Without their support, my Master's degree could not have been successfully completed.

Abstract

Participatory sensing systems need to gather information from a large number of participants. However, the openness of the system is a double-edged sword: by allowing practically any user to join, the system can be abused by an attacker who introduces a large number of virtual devices. This work proposes a hardened registration process for participatory sensing to raise the bar: registrations are screened through a number of defensive measures, towards rejecting spurious registrations that do not correspond to actual devices. This deprives an adversary from a relatively easy take-over and, at the same time, allows a flexible and open registration process. The defensive measures are incorporated in the participatory sensing application.

Keywords

Registration Process, Participatory Sensing, International Mobile Equipment Identity (IMEI), Android Rooting, Android Emulator, Mobile phone, CAPTCHA

Abstrakt

Deltagande avkännings system behöver samlas från ett stort antal aktörer. Systems öppenhet är dock en dubbelsidigt värd: Genom att låta alla praktiska användare deltagit, kan system utnyttja en av angripare som introducera ett stort antal virtuella enheter. I det här arbetet föreslå en härda registreringsprocess för deltagare att identifiera höjning av ribban: registrering screenas genom ett antal defensiva åtgärders, för att avvisa falska registreringar som inte motsvara aktuella enheter. Detta berövar en motståndare från en relativt lätt övertagande och ger samtidigt en flexibel och öppen registreringsprocess. De defensiva åtgärderna införlivas i deltagande avkännings applikation.

Nyckelord

Registreringsprocess, deltagande avkänning, International Mobil Equipment Identitet (IMEI), Android Rotning, Android Emulator, Mobil telefon, CAPTCHA

Contents

1 Introduction

Participatory sensing (PS) data collection takes advantage of the sensing, processing and storage resources in mobile phones to gain knowledge about the participants and, more so, their environment. The acquired information allows other participants to gain access to shared (contributed) data through a participatory sensing service enabled by a mobile application (that operates over wireless cellular or other networks). In this day and age, participatory sensing has a strong advantage due to the proliferation of smartphone users and advanced built-in smartphone sensors, which enable large-scale and diverse deployments (e.g. environmental [1],[2], disasters [3],[4] and traffic monitoring [5],[6]).

1.1 Problem definition & Thesis goal

To create a powerful participatory sensing service, dealing with security and privacy is important. The two major concerns are:

(a) Protecting the systems and ensuring the quality of the data.

In order to acquire accurate information in participatory sensing systems, broader participation gives better results. However, the openness of the participatory sensing systems might introduce the risk that any of the participants can be an adversary who pollutes and manipulates the collected data in the participatory sensing systems. Consequently, distorted information can lead other legitimate users to wrong decisions and the services or campaigns become useless.

It is also possible that registered users attack the participatory sensing systems. After they have registered, their mobile devices can be infected by a malware or a botnet and inject a large number of fault reports or submissions later. This implies that any misbehavior can be performed by both outsiders and intruders to the participatory sensing systems.

(b) Protecting the users' privacy while enabling incentive provision.

By submitting reports or issuing queries though the service provider in the participatory sensing system, the sensitive information of users (e.g. information about user's identity, location, lifestyle and habits) can be analysed and tracked. Moreover, it is not only the service provider that can learn this sensitive information but adversaries can do the same. This brings up privacy concerns to the users and they may refuse to use the service.

The mentioned concerns can be introduced due to the weakness or excessive information retrieving of the user registration process policy. The user registration process itself can be risky. On the one hand, participatory sensing thrives on broad participation; the easier the registration process is, the more likely users are to participate. On the other hand, the openness of the registration process, unless done correctly, can become a vulnerability. An adversary could create a small 'army' of virtual users (devices) and have them registered. Those devices could then be orchestrated to provide fake data and undermine the participatory sensing process. One possibility would be to spoof the International Mobile Equipment Identity (IMEI) [7], getting access with many accounts through rooting or using an IMEI spoofing application. As a result, the malicious participants could give fake information, perform a Distributed Denial-of-Service (DDoS) attack [8], or simply not contribute to the services. Therefore, dealing with the openness of the participatory sensing service through the registration process also promotes a good character of the service.

One defensive approach could be a strict registration process requiring, for example, to hand over the smartphone, corroborate the identity of the user, etc., having stepped into an authorized physical location and bootstrap credentials in place. This may be the only option for some applications, but it would be cumbersome.

There are two goals that this work aims to achieve:

1. Having the registration process such that the adversaries and Android botnets cannot infiltrate to degrade the quality of data collection in participatory sensing services or to stop the services while the registration

process does not force the users to provide their sensitive information that can reveal the identity or behavior of the users.

2. Having the registration process designed so that the users can register themselves from anywhere while the participatory sensing service is still protected.
3. Reducing the burden on participant monitoring for participatory sensing services by selecting only the participants that have a legitimate device and are located in a target area.

1.2 Limitations

1.2.1 The evasion techniques on a rooted smartphone

Even though the hardened registration process offers the rooting detection, there may be sophisticated evasion techniques that can evade the detection techniques that the hardened registration process offers due to the fact that rooting evasion and rooting detection are always on the race that favors the evader than the detector. However, we will discuss about the improvement of rooting detection in the future work section.

1.2.2 Accessibility of the disabled

The new CAPTCHA "MoCAPTCHA" and the gyroscope mini game that are developed in the hardened registration process cannot be accessible for users who are blind, visually impaired and hand impaired. The tasks from MoCAPTCHA and the gyroscope mini game require human visual recognition and hand movement to solve. Therefore, the hardened registration process can separate computers from humans, but also removes people with disabilities from performing the requested procedure.

1.2.3 The attack on participatory sensing service with real mobile devices

There is a possibility that the attack on participatory sensing service can be introduced by a large number of real mobile devices which are located in the target area. An adversary may recruit local volunteers or may offer local people with pay-off to use their real mobile devices for registration into the participatory sensing service to perform misbehaviors that degrade the quality of shared information in the service such as submitting a large number of bogus reports. If these real mobile devices are not modified, it cannot be rejected by the hardened registration process. Therefore, the hardened registration process is not a solution for this attack and we will discuss about other solutions which can solve this problem in the future work section.

1.2.4 Location spoofing with a fake cellular base station

There is a possibility that an adversary may use a fake cellular base station for jamming mobile phones' signal and providing fake cell-ID information to mobile phones. This fake cell-ID information will deceive the location inspector in the hardened registration process that the mobile phones are located in the target area which is actually not true.

Therefore, the hardened registration process is not a solution for jamming by a fake cellular base station.

1.3 Contribution & Outline

In this work, we take an alternative approach: we defend against an adversary that may try to swiftly register a large number of "users" it fully controls, by introducing a number of checks and controls. We integrate those in the participatory sensing mobile application. Although highly skilled adversaries may have ways to defeat such countermeasures, this first line of defence raises the bar and makes it significantly harder for such adversarial behavior to be effective. Furthermore, the proposed solution reduces significantly the burden on the participatory sensing service side because checks are to be performed on the user side. This has an added advantage: controls do not force disclosure of sensitive user information to the participatory sensing service.

This research presents an approach for the hardened registration process (HRP), designed to protect the participatory sensing services from abuse. Notably, among other controls and checks, the design brings forth the idea of a “mobile” CAPTCHA [9] tailored to smartphone platforms.

To show why we have to improve the registration process for participatory sensing and CAPTCHAs, we are going to discuss the background & related work in the next chapter which provides stories about the manner, problems, the existed solutions and the room for development regarding participatory sensing systems and CAPTCHAs. Then, to show how we improve these preliminary checks and control to raise a bar for the registration process, we introduce the system model & approach. Next, we show the result of our improvement in security and performance analysis section. At the end, conclusion, discussion and future work will be provided.

2 Background & Related work

2.1 Participatory sensing

Participatory sensing [10],[11],[12] is data collection which is begun and contributed by sensory information from people in the communities.

Although participatory sensing can leverage a variety of data collection devices such as local water stations and local weather stations, mobile devices are important equipments for engaging people in the communities to share information on their local environment.

The mobile devices such as smartphones have become popular in the present with around 1.54 billion worldwide smartphones subscribers [13]. It has been estimated that 1.1 trillion digital photos were taken worldwide in 2016, 81.3 percent of them were taken by smartphones and 2.5 trillion photos were shared or stored online globally [14]. This is a strong piece of evidence which proves that the smartphone is cutting-edge technology for collecting and sharing information.

The proliferation of smartphones along with the ubiquity of the broadband network infrastructure enables participatory sensing to appreciably expand the capabilities of gaining knowledge in the target areas where wireless sensor network (WSN) [15] applications have been employed during the last couple of decades. While the setting up of WSN may not be feasible for covering small details in the target areas, participatory sensing takes advantage of the deployment of built-in smartphone sensors to monitor this scenario. As a result, data collected by smartphone becomes extremely useful to fill the information in detail precisely. For instance, participants can report about their surrounding situation on the air pollution, temperature or noise level in real-time temperature or noise level. Similarly, the velocity of cars can be detected by drivers' smartphones which can later form the traffic conditions.

Project Budburst [16] is one of the participatory sensing projects that monitor phenological changes in the environment which inform the situation on global warming. Participants use their free mobile application and a smartphone camera to share photos of specific plants during different stages of their lifecycle (e.g., first bloom, first leaf). Conducting a global study of this magnitude would require vast resources and may not be suitable to deploy WSN, but through participatory sensing, Project Budburst has been able to collect nearly 14,000 data samples in five years, at minimal cost. The data from Project Budburst has shown shifts in phenological activity for stages of certain plant species.

A campaign provider such as the Project Budburst provider is not the only entity who drives the participatory sensing service, but there are also other entities that are needed for supporting the service as a participatory sensing infrastructure.

The typical participatory sensing infrastructure [12] involves (at least) the following parties:

1. “Mobile Nodes are the union of a carrier with a sensor installed on a mobile device. They provide reports and form the basis of any participatory application.”
2. “Queriers subscribe to information collected in a participatory sensing application and obtain corresponding reports.”
3. “Network Operators manage the network used to collect and deliver sensor measurements.”
4. “Service Providers act as intermediaries between Queriers and Mobile Nodes, in order to deliver reports of interest to Queriers. Queriers can subscribe to the appropriate Service Provider for one or more types of measurements.”

2.2 Security and privacy concerns in participatory sensing

From the previous section, the participatory sensing system shows that it provides an effective solution to gain knowledge about the participants and their environment. However, there are several security and privacy concerns that need to be addressed.

In the aspect of security, the issues of confidentiality and integrity need to be addressed. The adversaries may eavesdrop communications between entities in the participatory sensing system (e.g. Mobile Nodes and Service Providers or between Service Providers and Queriers) or exploit the vulnerability of the openness of the participatory sensing system by polluting the data collection to degrade the participatory sensing system. Cryptographic protection, authentication, authorization and state-of-the-art techniques such as SHIELD framework [17] that combine evidence handling and data-mining techniques to identify and filter out malicious contributions are effective solutions to ensure the confidentiality and integrity of the participatory sensing system.

In the aspect of privacy, the leakage of personal information of the participant has to be concerned. For example, the service provider who collects all data might learn a considerable amount of sensitive information about both Mobile Nodes and Queriers, and violate the privacy of their movements, interests, locations and habits. To safeguard the participants' privacy, some solutions have been introduced. For example, sensing data and location information from participants can be protected from the adversaries with a Conditional Random Field (CRF) [18] to model potential spatio-temporal correlations in PLP [19] or SPPEAR architecture [20] that separates processes and functions across entities to provide anonymity for participants.

To deal with security and privacy concerns in participatory sensing, developing the registration process can be one solution that can support the mentioned methodologies to strengthen security and preserve the participants' privacy through the registration process.

2.3 CAPTCHAs

The history of CAPTCHAs can be traced back to 1997 [21]. At that time, a web search engine, AltaVista was faced with a malicious bot on the Internet which was distorting the search engine's ranking algorithms by automatically submitting URLs to their web search engine.

AltaVista's scientists developed an algorithm that randomly generated an image of printed text for solving this problem. Computers couldn't recognize the image from the task, but humans could read the message the image contained and respond appropriately. Eventually, in April 2001, AltaVista's scientists issued a patent for the technology.

In 2003, Carnegie Mellon University and, a computer manufacturing company, IBM improved the algorithm and originated the term CAPTCHA which stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart.

2.3.1 How CAPTCHAs work

The general goal of a CAPTCHA is to defend an application against the automated actions of undesirable and malicious bot programs on the Internet. To achieve this goal, CAPTCHAs are designed to generate the tests that most humans can pass, but computer programs cannot. For example, when an image-based CAPTCHA sends a task to the automated bots, even though they can identify the existence of an image by reading source code, they cannot tell what the image depicts. However, some CAPTCHA tests are difficult to interpret, and users are usually given the option to request a new test.

2.3.2 Types of CAPTCHAs

The most common type of CAPTCHA is the text-based CAPTCHA. The text-based CAPTCHA requires the user to view a distorted string of alphanumeric characters in an image and enter the characters in an attached form. The text CAPTCHAs can be also rendered as audio recordings for the visually impaired.

Image-based CAPTCHAs, which are also commonly used, ask users to identify a subset of images within a larger set of images. For instance, the user may be given a set of images and asked to click on all the ones that have cars in them.

Other types of CAPTCHAs include:

- Math CAPTCHAs - require the user to solve a basic math problem, such as adding or subtracting two numbers.
- reCAPTCHA [22] - requires the users to check a box.
- NuCAPTCHA [23]- require the users to type a word that they see in an animation. The animation consist of alphanumeric characters that overlap as they swing independently left to right.

Even though there are many types of CAPTCHAs that can be solutions for a computer platform, these CAPTCHAs might not have been designed for a mobile platform and sometimes, for users, solving these CAPTCHAs on their mobiles is limited. Thus, there is a gap of development for CAPTCHAs on a mobile platform since it is needed for mobile-based services such as participatory sensing services.

2.4 Registration process

There are some existing registration processes that try to cope with participant selection in the participatory sensing systems. The work in [24] handles the problems arising from the large scale of the mobile participatory sensing systems. To reduce the number of devices involved, the selection count on the probability of other devices being present at the locations of their expected path is based on a realistic human mobility model called Truncated Lévy Walk. If the necessary information of a target area has been fulfilled by other sensors, the participatory sensing service will not allow the new participant to register her device to the service. However, the methodology does not pay attention to the intervention of botnets or unusual devices such as rooted devices and emulators that open the possibility to overwhelm the service by malicious manners.

Participant selection in [25] deals with personal information, personal activities and social activities which may lead to privacy concerns. The reputation system that has been introduced in this research uses identifying mobile agents to help the participatory sensing service screen participants who are not likely to contribute to the service. However, by letting unusual devices pass through the recruitment, the adversaries have the possibility to harm the service before their mobile agents can find any misbehavior.

The recruitment in [26], which is similar to our research, introduces a location mechanism to detect whether the participant is located in the target area or not. However, there is no rooted device or emulator detection. Therefore, there is a possibility that the location can be spoofed by those unusual devices.

There is an existing work on the mobile terminal public service [27] that offers an authentication scheme for a voting system by using the International Mobile Subscriber Identity (IMSI) [28] and One Time Password (OTP) [29] through e-mail or mobile phone. The security of this scheme can be intruded upon since IMSI can be spoofed by rooted devices or emulators. Moreover, a new e-mail account or a mobile phone number can be simply created.

3 System model & Approach

The goal of this work is to protect participatory sensing services from adversaries who aim to degrade and to occupy the participatory sensing services. The method that this work offers is to inspect devices that try to register itself into the participatory sensing services with checks and controls which are implemented in an Android application for whoever would like to join the participatory sensing services. Furthermore, the checks and controls should not force users to disclosure their sensitive information to the participatory sensing services due to privacy concerns.

3.1 Adversarial tools

In order to degrade and occupy a participatory sensing service, an adversary can use tools that can give a superuser privilege which allows the adversary to acquire full control and free customization of the device or can use malicious applications for manipulating the information of a smartphone to create a number of fake accounts.

For gaining the superuser privilege, the adversary can exploit a rooted device or emulators for Android phones. After the adversary has gained the superuser privilege, she can spoof the Global Navigation Satellite Systems (GNSS) [30] location or IMEI number to create a number of fake accounts. With a number of fake accounts, the adversary can provide a mass of fake information to degrade the trust in the participatory sensing service or use botnets to perform a DDoS attack by injecting a number of requests to stop the participatory sensing service. In addition, the adversary can apply techniques in [31] to avoid the detection of the rooted smartphone or the emulators.

Another approach to create a number of fake accounts is to use malicious applications that can be downloaded from Android markets which can spoof the GNSS location or IMEI number without using the rooted smartphone or the emulators.

All of these adversarial tools are used for performance testing in the hardened registration process from which the result of the testing will be illustrated later in the evaluation section.

3.2 Adversarial classification

The adversaries are classified by the performance of the adversarial tools that they use and the level of knowledge that they must possess for attacking the participatory sensing service.

Level 0 - Legitimate user: A legitimate user is the user who uses an untampered smartphone, does not use an emulator or a rooted smartphone which introduces malicious actions and does not install any malicious applications.

Level 1 - Beginner adversary: A beginner adversary is the adversary who installs malicious applications that can be easily found in application markets to spoof the information of the smartphone such as the GNSS location or IMEI number without knowledge on how the location mechanism and IMEI number work.

Level 2 - Intermediate adversary: An intermediate adversary is the adversary who uses a rooted smartphone or uses an emulator to gain a superuser privilege. The adversary manipulates the information of the smartphone such as the GNSS location or IMEI number without knowledge on how the location mechanism and IMEI number work. The adversary does not know how to evade the detection of the rooted smartphone or the emulator.

Level 3 - Advanced adversary: An advanced adversary is the adversary who uses a rooted smartphone or uses an emulator to gain a superuser privilege. Although the adversary knows how to evade the detection of the rooted smartphone or the emulator, the adversary manipulates the information of the smartphone such as the GNSS location or IMEI number without knowledge on how the location mechanism and IMEI number work.

Level 4 - Professional adversary: A professional adversary is the adversary who uses a rooted smartphone or uses an emulator to gain a superuser privilege. The adversary knows how to evade the detection of the rooted smartphone or the emulator. The adversary can also propagate botnets through smartphone applications or malicious pop-up advertisements on websites and employs those botnets to perform the DDoS attack or undertake actions for other malicious purposes. Furthermore, the adversary manipulates the information of a device such as the GNSS location or IMEI number with knowledge on how the location mechanism and IMEI number work and reports fake sensory information to mislead other users in the participatory sensing services.

Level	Types	Actions	Adversarial tools
0	Legitimate user	- uses an untampered smartphone. - does not install any malicious applications.	
1	Beginner adversary	- installs malicious applications to spoof the GNSS location or IMEI number. - does not have knowledge on how the location mechanism and IMEI number work.	- Fake GPS Location - Fake GPS GO Location Spoofer - XPOSED IMEI Changer Pro - MTK Engineering Mode
2	Intermediate adversary	- gains a superuser privilege to spoof the GNSS location or IMEI number. - does not have knowledge on how the location mechanism and IMEI number work. - does not apply the rooting or emulating evasion techniques.	- A rooted smartphone - Android Studio Emulator - BlueStacks 3 - Nox Emulator - MEmu Emulator
3	Advanced adversary	- applies the rooting evasion applications. - does not have knowledge on how the location mechanism and IMEI number work.	- A rooted smartphone - Hide my Root - Hide Rooting Lite
4	Professional adversary	- may use a customized emulator which can emulate the Bluetooth function. - applies the rooting evasion applications. - may apply sophisticated rooting evasion techniques. - has knowledge on how the location mechanism and IMEI number work. - uses botnets. - prepares sensor data.	- Customized emulator - Sophisticated rooting evasion techniques - Android botnets - Prepared sensor datasets

Table 1: Adversary model

3.3 Hardened registration process

The hardened registration process, as illustrated in the figure 3.1, offers a process that can protect participatory sensing services from malicious behaviors.

The adversary could mislead the participatory sensing system by seemingly registering a large set of devices in the area of interest, while, in reality these could be virtual entities, exhibiting arbitrary or malicious behavior. These could be, of course, rooted smartphones, a smartphone emulator or a botnet. This way, the

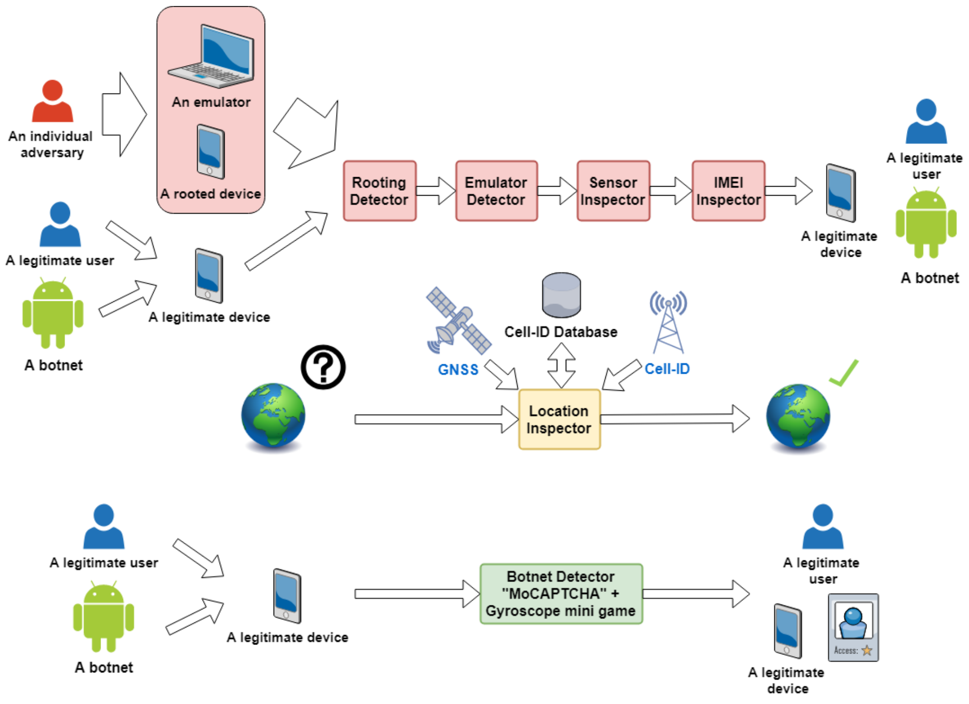


Figure 3.1: Illustration of the HRP for PS services.

adversary could overwhelm the participatory sensing service by faking sensor data and smartphone locations. For example, the sybil attack on Crowdsourced Mobile Mapping Services [32] has introduced and performed the sybil attack on Waze, a popular crowdsourced map service, by using an emulator to create the number of virtual devices and to use these virtual devices for reporting false congestion and accidents, and automatically rerouting user traffic.

From the example case, the hardened registration process defends against the attack with checks and controls, which are rooting detection, emulator detection, sensor inspector, IMEI inspector, location inspector and botnet detector.

Checks and controls are complementary and reinforce one another. In the hardened registration process, the rooting and emulator detectors are designed to support the sensor, location and IMEI inspectors by reducing choices of the adversary to exploit a superuser privilege for sensor data, location and IMEI spoofing.

Rooting detector: The Android rooting process essentially converts Android smartphone normal user credentials and permissions into those of a superuser, allowing full control and free customization of the smartphone. This introduces risks allowing adversaries to manipulate smartphone information that is relevant to the participatory sensing service. However, effective techniques [31] can be applied to detect a rooted device through the application program interface (API); they can be incorporated into the hardened registration process in order to, for example, check directory permissions for writability and check the existing superuser path.

Emulator detector: Launching an emulator allows adversaries to manipulate smartphone information and adversely affect participatory sensing services. In response, the hardened registration process applies techniques [33] to detect the emulation behavior. Detecting the existence of Bluetooth is a very effective way to do so because Bluetooth is not present in emulators. The support of emulator detection through the Android API and inspection of Android system properties makes the emulator detector more effective.

Sensor inspector: Sensors in smartphones are important for providing information to participatory sensing services. Thus, only smartphones with good-condition sensors should be given access to the services. Sensor condition, calibration and diagnostics can leverage the SensorManager API.

IMEI inspector: An IMEI number can be spoofed; thus, the IMEI inspector compares the smartphone model information that can be retrieved from the Android API against the information from the Type Allocation Code (TAC) [34] database to reject mismatches. Duplicated IMEI number can be detected as well. In the case that the IMEI inspector can gain knowledge about the registered IMEI numbers, it can check the IMEI number of a smartphone with the registered user database to ensure that the IMEI number does not exist. If an existing IMEI number does exist in the database, the newly registered IMEI number will be rejected from the system.

Location inspector: The hardened registration process allows participants to access participatory sensing services if their smartphone is located in a target area. However, the location of a smartphone may be manipulated in order to be eligible by the participatory sensing for a given task. Thus, before granting access, the hardened registration process confirms the smartphone location, e.g., combining information from surrounding cell towers and the GNSS location.

Botnet detector: The botnet detector attempts to determine whether the apparent smartphone is part of a botnet or not; or, inversely, whether it is an actual human being with an actual phone seeking to join the participatory sensing system. The hardened registration process offers the smartphone-friendly MoCAPTCHA; it integrates multi-touching, an accelerometer sensor and gravity sensor interaction along with human visual recognition. Our MoCAPTCHA requires the participant to enact gestures that correspond to the provided tasks. An additional step to strengthen the security is introduced by prompting the participant to solve a gyroscope mini game which is designed in a way that the submitted sensor data cannot be prepared beforehand.

In order to have a number of fake accounts and be able to access the participatory sensing service from anywhere, the adversary has to spoof IMEI number and mock the location with some adversarial tools, and later provide fake information (e.g., fake sensor data and fake reports). Therefore, the hardened registration process not only stops those malicious actions by an IMEI inspector, sensor inspector and a location inspector, but also blocks the adversarial tools that the adversary might use for the malicious actions by the support of the rooting detector, emulator detector and botnet detector even before the adversary starts to perform the attacking.

3.4 System implementation

All checks and controls in the hardened registration process are implemented as an Android application. The application will be installed in a user's device if they would like to get access to the registration process. The characteristic information of the device will be exchanged between the application and databases through a web service to prove whether the device has the right to access the participatory sensing service or not.

3.4.1 Android platform

This research uses Android Studio 3.0.1 to create the secured registration process application for Android OS version 4.0 (Ice Cream Sandwich) or the higher which supports a framework API level 14. The application works as a user interface for participants to register their devices and to prove their rights for getting access to a participatory sensing service. The application sends and receives information between the participants and a web service by using JSON data.

3.4.2 Web service

This research uses Apache Tomcat with Spring boot for providing web services in the Spring Framework [35]. This platform works on Amazon Web Server to relay the information from databases to the related queries from the hardened registration process application.

3.4.3 Databases

Databases support the IMEI and location inspector in the hardened registration process to provide all information that the application needs.

The databases are implemented by MongoDB. The application can retrieve all information from the databases through the Tomcat web service.

Table 2 shows the information that the application can get from the devices and the information that returns from each type of database.

A type of the databases	**The received information from the devices**	**The returned information from the database**
IMEI database	TAC	Brand name, device name and model name
Cell ID datebase	MNC, MCC, CID and LAC	Latitude and longitude
**Registered user database	IMEI	Matching status

Table 2: The received information that from the devices and the returned information from each type of database.

**Registered user database only exists for the participatory sensing service that provides the information about the registered IMEI numbers to the hardened registration process.

3.5 Checks and controls implementations

3.5.1 Rooting detector

The Android rooting process will switch an Android smartphone user's credentials and permissions from a normal user to a superuser which allows the user to fully control and freely customize her own devices. The root privileges from gaining root access in the Android smartphone attracts some Android smartphone users to perform Android rooting [31] for getting rid of bloatware, performing full system backup and overclocking the smartphone CPU to increase its performance. However, the Android rooting introduces some security risks to the participatory sensing services by creating vulnerabilities in the smartphone to malware infection and gaining opportunities for the adversaries to manipulate smartphone information and occupy the participatory-sensing services. Thus, rooting detection in the hardened registration process is truly needed. Android Rooting: An Arms Race between Evasion and Detection by Nguyen-Vu et al. [33] conducted various rooting detection techniques. The rooting detector (see appendix A) implements 4 techniques to detect rooting behavior in the hardened registration process.

- Using getPackageInfo to retrieve information about an installed package

  This technique checks application packages with a list of packages that are related to rooting behaviors. The getPackageInfo methods will return overall information about installed packages/applications in a smartphone. We search for suspected packages/applications information, such as

  - de.robv.adnroid.xposed.installer
  - com.devadvance.rootcloakplus
  - com.noshufou.android.su
  - eu.chainfire.supersu
  - com.formyhm.hideroot

  By using this technique, we can update the list of dangerous packages if new rooting techniques are published in Android markets.

- Checking directory permissions

  This technique checks the directory paths via “mount” adb shell command that a normal user must not have permission to write anything in these following directories:

  - /system
  - /system/bin
  - /system/sbin
  - /system/xbin
  - /vendor/bin
  - /sbin
  - /etc

  If these directories are writable, the device will be rejected as the user has the superuser privilege.

- Checking Android OS properties

  By checking Android system properties such as ro.kernel.qemu, ro.secure and ro.debuggable, we may detect rooting behavior. If the value of ro.secure is “0”, or the value of ro.kernel.qemu is “1”, ADB shell runs as rooted and that means the device is a rooted device. Another approach is to check ro.debuggable and service.adb.root. If these are both “1”, privileges will not be dropped even if ro.secure is set to “1”. There is no Android API for checking the status of these properties. We have to use “getprop” adb shell command to inspect these properties.

- Searching for SU path in the smartphone

  This technique searches for the SU path in the directories where “su” usually exists in when the user performs rooting in these following directories:

  – /data/local/bin

  – /su/bin

  – /system/bin/

  – /system/xbin

  – /system/sd/xbin/

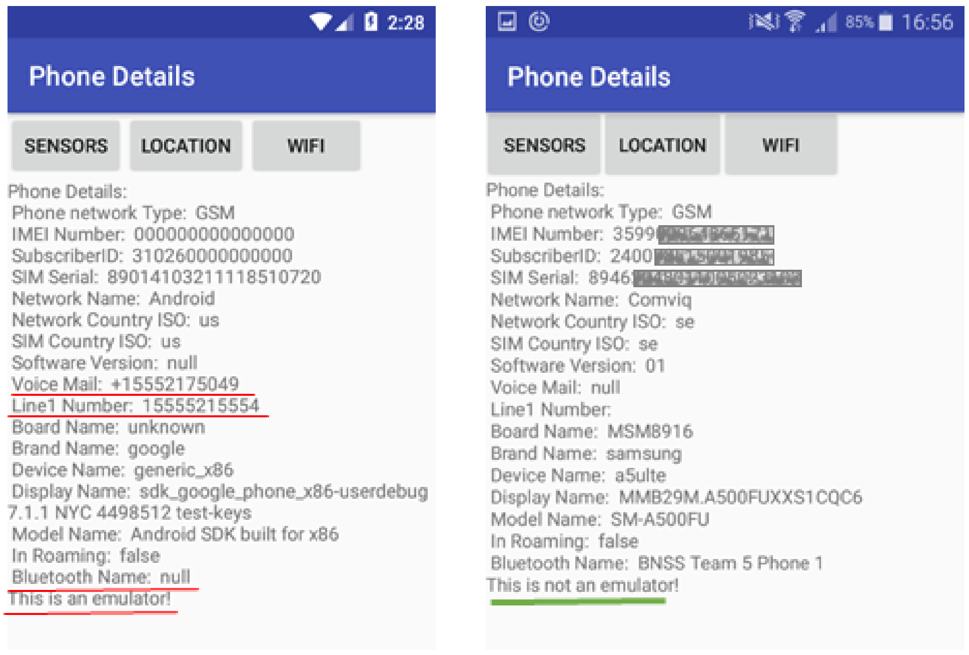
  – /data

  – /dev

  To implement this, we search for “su” by using file.exists() method to check for the SU binary file in the suspected directories.

3.5.2 Emulator detector

Allowing an Android emulator to access the participatory sensing services may grant opportunities to the adversaries for mocking the location, manipulating sensors information, performing DDoS attack and spoofing unique identifiers. These problems open a possibility for the adversaries to occupy the services with a large number of devices. The hardened registration process offers



```
Phone Details:
 Phone network Type: GSM
 IMEI Number: 000000000000000
 SubscriberID: 310260000000000
 SIM Serial: 89014103211118510720
 Network Name: Android
 Network Country ISO: us
 SIM Country ISO: us
 Software Version: null
 Voice Mail: +15552175049
 Line1 Number: 15555215554
 Board Name: unknown
 Brand Name: google
 Device Name: generic_x86
 Display Name: sdk_google_phone_x86-userdebug
7.1.1 NYC 4498512 test-keys
 Model Name: Android SDK built for x86
 In Roaming: false
 Bluetooth Name: null
This is an emulator!
```

```
Phone Details:
 Phone network Type: GSM
 IMEI Number: 3599[redacted]
 SubscriberID: 2400[redacted]
 SIM Serial: 8946[redacted]
 Network Name: Comviq
 Network Country ISO: se
 SIM Country ISO: se
 Software Version: 01
 Voice Mail: null
 Line1 Number:
 Board Name: MSM8916
 Brand Name: samsung
 Device Name: a5ulte
 Display Name: MMB29M.A500FUXXS1CQC6
 Model Name: SM-A500FU
 In Roaming: false
 Bluetooth Name: BNSS Team 5 Phone 1
This is not an emulator!
```

Figure 3.2: Illustration of the emulator detection.

some techniques to detect whether the signing-up smartphone is launched from the Android emulator or not. Evading Android runtime analysis via sandbox detection by T. Vidas and N. Christin [36] conducted various emulation detection techniques which can be applied to the emulator detector (see appendix B) in the hardened registration process as illustrated in figure 3.2.

- Detecting emulation through Bluetooth

  As mentioned that Bluetooth is a device which can be found in a smartphone, but cannot be found in emulators itself, we are going to use the "BluetoothAdapter.getDefaultAdapter()" method and check the returned value from the device. If a valid handle is returned, we can prove that the signing-up smartphone is a physical device.

- Detecting emulation through the Android API

  By using the Android API method, the emulation behaviors can be detected through returned values on TelephonyManager.

  When checking the voice mail number and line1 number, some emulators return +15552175049 for the voice mail number and 155552155xx for the line1 number. xx indicates a small range of ports for a particular emulator instance as obtained by the Android Debug Bridge (ADB). Emulator instances begin at 54 and will always be an even number between 54 and 84.

- Checking Android OS properties

  We apply the same methodology in the rooting detection on checking Android OS system properties. In other words, we check Android system properties such as ro.kernel.qemu, ro.secure, ro.debuggable and service.adb.root via "getprop" adb shell command to inspect these properties.

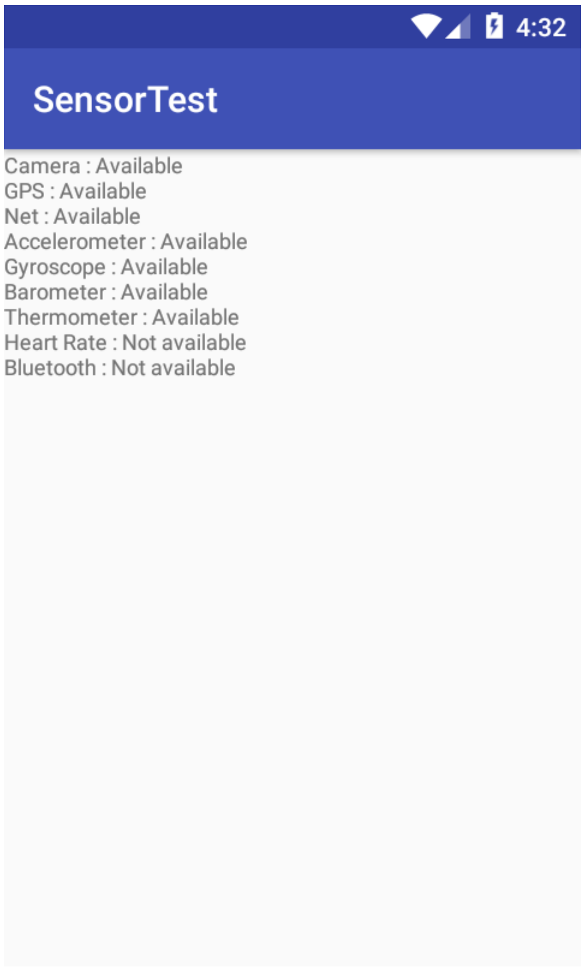
3.5.3 Sensor inspector

We are now going to inspect the availability of the basic sensors in the signing-up smartphone on whether the basic sensors are in working condition or not (see appendix C). The sensors that have to be inspected are the accelerometer, gyroscope, magnetometer, camera, multi-touching, GPS receiver, network-based geolocation system and WiFi. The Android API method that is used for the inspection here is "manager.hasSystemFeature" as illustrated in figure 3.3.

There is another support for detecting emulation behavior by checking the name of sensors. We use getSensorList API method to check the names of sensors in the device as illustrated in figure 3.4. If we find "Goldfish" in the returned value of sensorText (the names of sensors), the device is launched from an emulator.

3.5.4 IMEI inspector

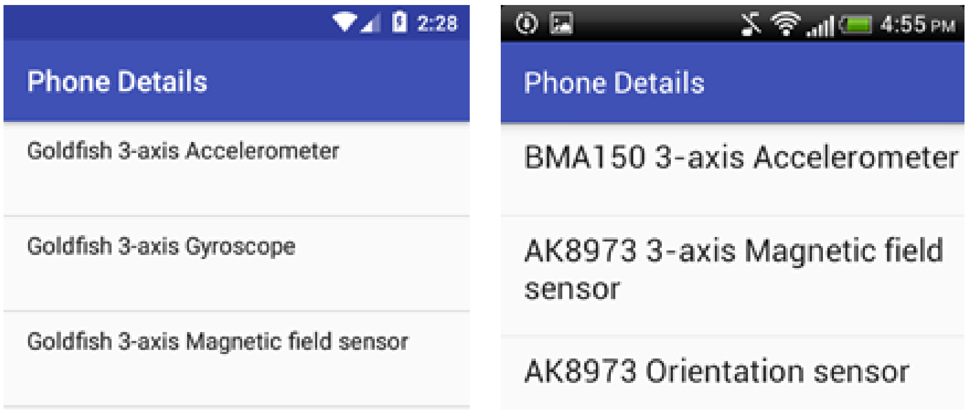
The International Mobile Equipment Identity (IMEI) is a unique number for mobile phones. We can get this number from the Android API method android.telephony.TelephonyManager.getDeviceId(). To protect the participatory sensing services from the adversaries who try to occupy the services, the IMEI inspector (see appendix D) will collect the IMEI number in a database and compare it with a new signing-up smartphone. If the new IMEI number


4:32

SensorTest

Camera : Available
GPS : Available
Net : Available
Accelerometer : Available
Gyroscope : Available
Barometer : Available
Thermometer : Available
Heart Rate : Not available
Bluetooth : Not available

Figure 3.3: Illustration of the sensor inspector.


2:28

Phone Details

Goldfish 3-axis Accelerometer

Goldfish 3-axis Gyroscope

Goldfish 3-axis Magnetic field sensor

4:55 PM

Phone Details

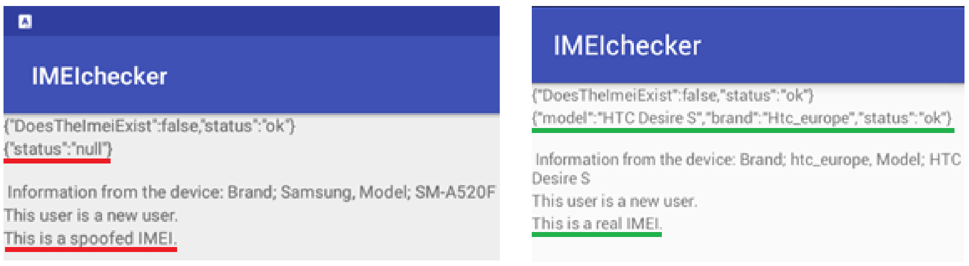
BMA150 3-axis Accelerometer

AK8973 3-axis Magnetic field sensor

AK8973 Orientation sensor

Figure 3.4: Sensors' names that are retrieved from an emulator (left) and a legitimate smartphone (right).

matches with the collected IMEI number, it indicates that the participant is likely to occupy the services and the hardened registration process will reject the registration request from the process. Note that the hardened registration process



```
{"DoesTheImeiExist":false,"status":"ok"}
{"status":"null"}

 Information from the device: Brand; Samsung, Model; SM-A520F
This user is a new user.
This is a spoofed IMEI.
```

```
{"DoesTheImeiExist":false,"status":"ok"}
{"model":"HTC Desire S","brand":"Htc_europe","status":"ok"}

 Information from the device: Brand; htc_europe, Model; HTC
Desire S
This user is a new user.
This is a real IMEI.
```

Figure 3.5: Illustration of the IMEI inspector.

API methods	**Information**
Build.BRAND	The brand of the device
Build.DEVICE	The name of the industrial design
Build.MODEL	The end-user-visible name for the end product
TelephonyManager.getDeviceId()	The IMEI number of the device

Table 3: The information from Build and TelephonyManager API methods for the IMEI inspector.

may be applied with a multi-entities participatory sensing architecture and cannot access the registered user database (table 4) due to anonymity preserving. In this case, the IMEI inspector may encrypt a message that encloses an IMEI number and a nonce, attach it with a user certificate and let the entity who has the information about registered IMEI numbers decrypt the message and check for the IMEI number duplication to accept or revoke the user certificate later. In addition, there is a possibility that the IMEI number is spoofed. Thus, the IMEI checker will compare the model information of the signing-up smartphone which is obtained from the Android API Build method (table 3) with the model information of the smartphone from the TAC database (table 5). The Type Allocation Code is a part of the IMEI number which can identify a particular characteristic of a smartphone including its manufacturer, model code and series code. TAC is the initial eight-digit portion of the 15-digit IMEI which the IMEI inspector can compare with the acquired IMEI number from the TAC database and procure the model information of the smartphone. If the information from the signing-up smartphone and TAC database is mismatched or unavailable, it indicates that the participant has manipulated the IMEI number and the hardened registration process will reject the registration request from the process as illustrated in figure 3.5.

User’s IMEI number	Username	Token
3518270100893231	Nicky	Osdfp4344e
3524420198443572	Pakoo	Kfds62kds1
8659440254600032	David	Caws9ad780

Table 4: Registered user database sample.

TAC	BRAND	BOARD	MODEL
35960200	Samsung	SGH	SGH-T209
35985400	Htc	FORE100	FORE100
86993000	Huawei	E398	E398

Table 5: TAC database sample.

3.5.5 Location inspector

The hardened registration process allows the participants to access the participatory sensing services if the signing-up smartphone is located in the target area of services. However, there is a possibility that the location of a smartphone is manipulated to gain access from the services. Thus, before checking the smartphone location, the hardened registration process has to prove whether the smartphone location is manipulated or not (see appendix E). Location manipulation can be conducted by some Android applications which can be easily found in application markets. Note that even though the hardened registration process has already filtered out the rooted Android smartphone, these Android applications can still manipulate the location without rooting. In order to prove that the smartphone location is not manipulated, the hardened registration process will mainly track Cell-ID from the Android smartphone which collects the location data from the surrounding cell towers and compares the result with the GNSS location. If cellular and GNSS locations confirm the same result, the smartphone location will be defined as a true location. Detecting GNSS Attacks on Smartphones by Shokouh. [37] conducted a methodology to detect the GNSS location of an Android smartphone with the support of surrounding WiFi and cell towers. The methodology can be applied to the hardened registration process for finding the true location of the participant’s Android smartphone.

API methods	**Information**
GsmCellLocation. getCid	Cell ID
GsmCellLocation. getLac	Location Area Code
TelephonyManager.getNetworkOperator()	MCC+MNC

Table 6: The information from Android API methods for retrieving cellular network location.

CID	**MCC**	**MNC**	**Area**	**Latitude**	**Longitude**
58667	240	7	61970	59.07173	11.258801
4226	240	1	2216	59.072831	11.248179
1216	240	1	12211	59.080451	11.250015
27263	240	1	2216	59.083377	11.251812

Table 7: Cell ID database sample.

- Retrieving the cellular network location

  To retrieve cellular network location from the device, we implemented Android API methods that are shown in table 6.

  After we retrieved the cellular network location from the device, we checked the information with the Cell ID database (table 7) to pinpoint a position (latitude and longitude).

- Retrieving the GNSS location

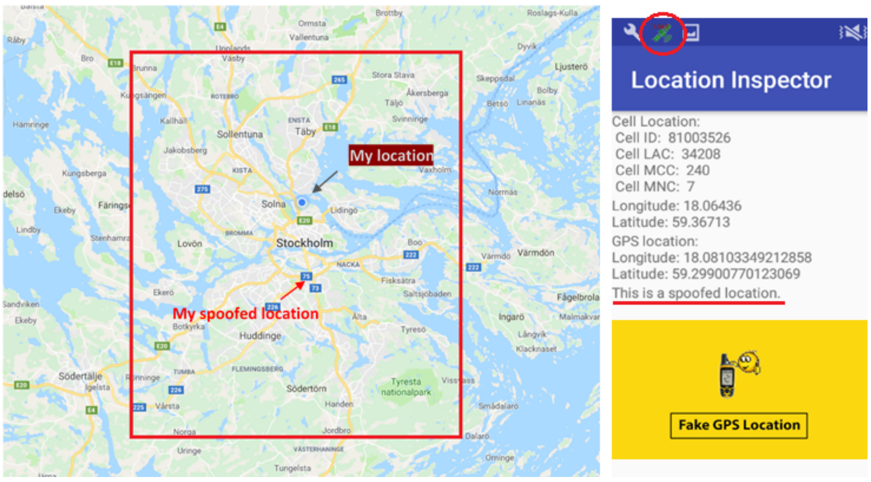
  To retrieve the GNSS location from the device, we implemented Android API methods as shown in table 8. To acquire these information, ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION permissions are needed.

- Decision making

  After we retrieved all types of location data, we compared the Cell ID location with the GNSS location. If the participant's GNSS location is not located

API methods	**Information**
location.getLongitude()	Longitude
location.getLatitude()	Latitude

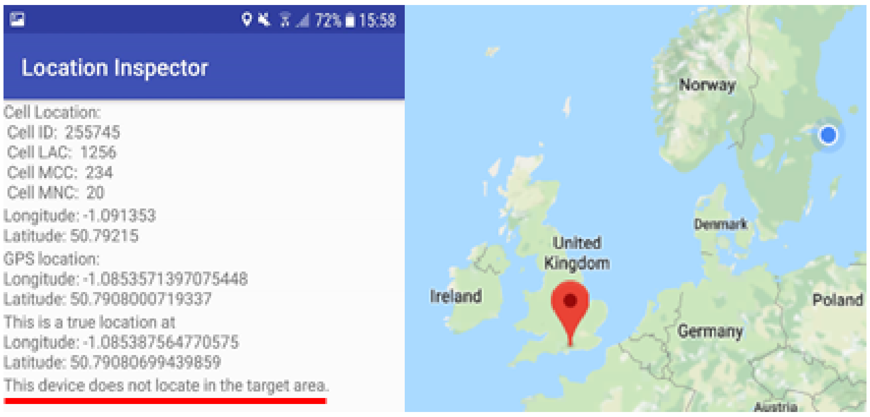
Table 8: The information from Android API methods for retrieving the GNSS location.



Location Inspector

Cell Location:
 Cell ID: 81003526
 Cell LAC: 34208
 Cell MCC: 240
 Cell MNC: 7
Longitude: 18.06436
Latitude: 59.36713
GPS location:
Longitude: 18.08103349212858
Latitude: 59.29900770123069
This is a spoofed location.

Figure 3.6: Illustration of the spoof location.



Location Inspector

Cell Location:
 Cell ID: 255745
 Cell LAC: 1256
 Cell MCC: 234
 Cell MNC: 20
Longitude: -1.091353
Latitude: 50.79215
GPS location:
Longitude: -1.0853571397075448
Latitude: 50.7908000719337
This is a true location at
Longitude: -1.085387564770575
Latitude: 50.79080699439859
This device does not locate in the target area.

Figure 3.7: Illustration of the location that is located outside the target area.

within a radius of 2 kilometres as compared to the Cell ID location, as illustrated in figure 3.6, the participant will be rejected from the hardened registration process since we consider that this is a spoofed location. If the participant's location fulfils the condition above, we will set the participant's GNSS location as the true location.

- Checking the location with the target area boundary

  In this research, we defined Stockholm city as a target area. Thus, we limited the latitude and longitude values to cover an area that is 20 kilometres from Gustav Adolfs torg as the Stockholm city boundary. We compared the participant's true location with this boundary. If the the true location is not located in the target area, the participant will be rejected from the hardened registration process as illustrated in figure 3.7.

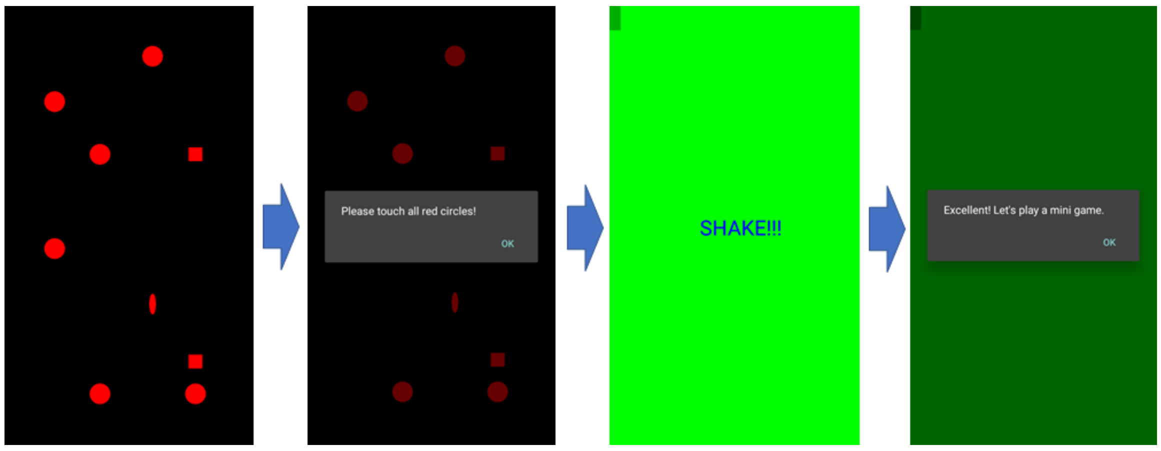

Please touch all red circles!
OK
SHAKE!!!
Excellent! Let's play a mini game.
OK

Figure 3.8: Illustration of the MoCAPTCHA.

3.5.6 Botnet detector

- MoCAPTCHA

  We offer the smartphone-friendly CAPTCHA "MoCAPTCHA" which collaborates with multi-touching and shaking tasks for the participants to provide their screen touching positions and motion sensors datasets related to the corresponding puzzles. The MoCAPTCHA (see appendix F) randomly generates the number and location of circles and other geometric shapes on the screen. When a participant touches all the circles that appear on the screen, the MoCAPTCHA will notify them to shake their smartphone. Shaking detection monitors the change on accelerometer data to capture the shaking that is made by human intention from amplitude and time on each dimension of the accelerometer. If the participant achieves the task, MoCAPTCHA will generate another puzzle randomly for the participant to solve as illustrated in figure 3.8. While the participant is solving the puzzles in the MoCAPTCHA, the gyroscope data is monitored by taking and comparing the gyroscope data samples to capture the tilt angles on smartphones that are made by human movement. The MoCAPTCHA distinguishes the difference between human and botnet (also actions that are made by the emulators) based on these capturing. Therefore, if botnets (on even emulators) have been used to access the participatory sensing service, the MoCAPTCHA will reject them from the hardened registration process.

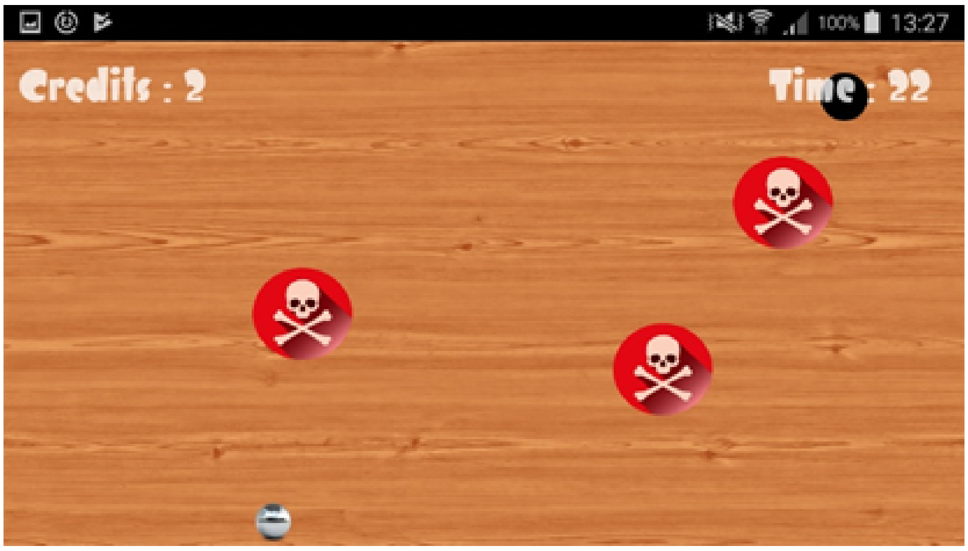


Figure 3.9: Illustration of the gyroscope mini game.

- A gyroscope mini game

  To strengthen security in the botnet proving implementation, we have to ensure that the sensors' data that are submitted cannot be prepared. Thus, we send a gyroscope mini game to the participant to solve as illustrated in figure 3.9. The goal of this mini game is letting the participant control a ball by tilting her smartphone and rolling it into a goal without touching any traps. The gyroscope mini game monitors the accelerometer data to define the speed of the ball and monitors the gyroscope data to define the direction of the ball that is bound in the game screen. The positions of a goal (a black hole) and traps (red skulls) have been set on the screen randomly. The participant will have three attempts within the limited time to win this game. If the participant cannot win the game, she will be suspected that she has prepared the sensor data and will be rejected from the hardened registration process.

4 Security & performance analysis

The security analysis of the hardened registration process is separated into each level of the adversarial classification that starts from the beginner adversary to the professional adversary.

Each level describes adversarial tools, attacking methods and the result of the attacking.

Then, we discuss about the time consumption and the reviews from users who try the trial version of the botnet detector.

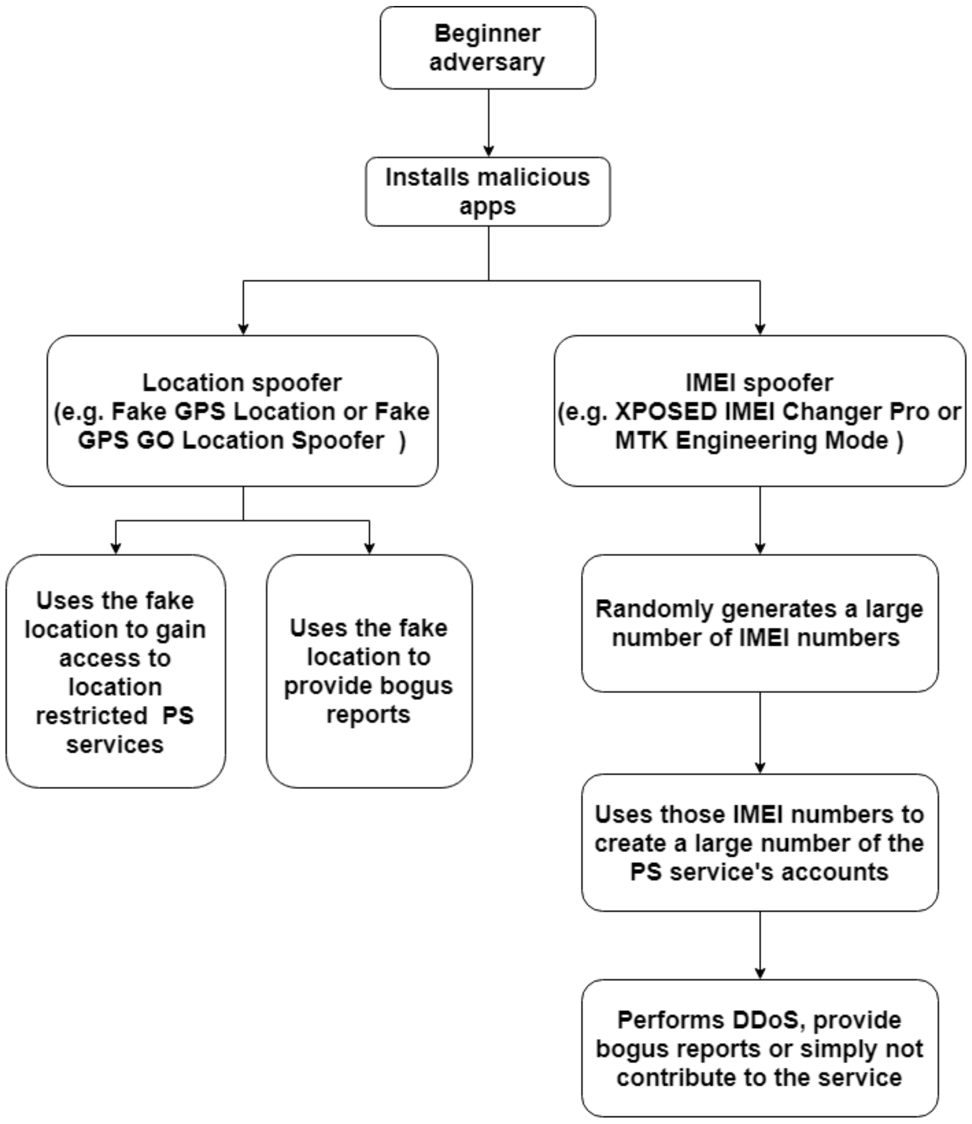
4.1 Attacking by the beginner adversary

4.1.1 Adversarial tools

On this level, the beginner adversary only exploits ready-to-use malicious applications to attack a participatory sensing service. The malicious applications that are used in this work are Fake GPS Location [38], Fake GPS GO Location Spoofer [39], XPOSED IMEI Changer Pro [40] and MTK Engineering Mode [41]. These have been installed on HTC Desire S.

4.1.2 Attacking methods

1. The adversary uses Fake GPS Location or Fake GPS GO Location Spoofer to spoof the GNSS location to gain access to a participatory sensing service from the location outside the interested area. When the adversary can gain access to the participatory sensing service, she can provide fake information to degrade the accuracy of the local information in the participatory sensing service.
2. The adversary uses XPOSED IMEI Changer Pro or MTK Engineering Mode to randomly spoof the IMEI number to gain access to a participatory sensing service. The adversary can create a number of fake accounts to



Beginner adversary
Installs malicious apps
Location spoofer (e.g. Fake GPS Location or Fake GPS GO Location Spoofer)
IMEI spoofer (e.g. XPOSED IMEI Changer Pro or MTK Engineering Mode)
Uses the fake location to gain access to location restricted PS services
Uses the fake location to provide bogus reports
Randomly generates a large number of IMEI numbers
Uses those IMEI numbers to create a large number of the PS service's accounts
Performs DDoS, provide bogus reports or simply not contribute to the service

Figure 4.1: Attack procedures by beginner adversary

perform a Distributed Denial-of-Service (DDoS) attack, provide a mass of fake information or simply not contribute to the services.

4.1.3 The result of the attack on the location

By using Fake GPS Location or Fake GPS GO Location Spoofer, the adversary can easily choose the location where the participatory sensing service accepts the participant to gain access for joining the service via interface of the applications. In this case, the true location of the device (HTC Desire S) is Forskarbacken in Stockholm, but it has been spoofed to be located in Gullmarsplan in Stockholm which is about 8 kilometres away from the true location. At the time that the hardened registration process inspected the location of the adversary, it found that this location had been spoofed since the GNSS location and the cellular location do not coincide. Therefore, the adversary is rejected since location spoofing has been detected by the location inspector.

4.1.4 The result of the attack on the IMEI number

By using XPOSED IMEI Changer Pro or MTK Engineering Mode, the adversary can easily choose any IMEI number that is different from the IMEI number which was used to gain access to the service via interface of the applications. In this case, these IMEI spoofing applications cannot perform correctly. There is no change after applying any of the IMEI numbers. Although this work has been used in other IMEI spoofing applications in Android markets, none of these applications work correctly. Consequently, to spoof the IMEI number, the adversary may have to gain a superuser privilege via a rooted device or an emulator.

4.2 Attacking by the intermediate adversary

4.2.1 Adversarial tools

Taking advantage of the superuser privilege, the adversary exploits the use of a rooted device or an emulator to occupy the participatory sensing service. This test changes a HTC Desire S from a legitimate smartphone to a rooted smartphone and uses various emulators – BlueStack 3 emulator [42], Android Studio emulator [43], MEmu emulator [44] and Nox emulator [45].

4.2.2 Attacking methods

1. The adversary uses an emulator to spoof the GNSS location to gain access to a participatory sensing service from the location outside of the interested area without using any evasion techniques. When the adversary can gain access to the participatory sensing service, she can provide fake information

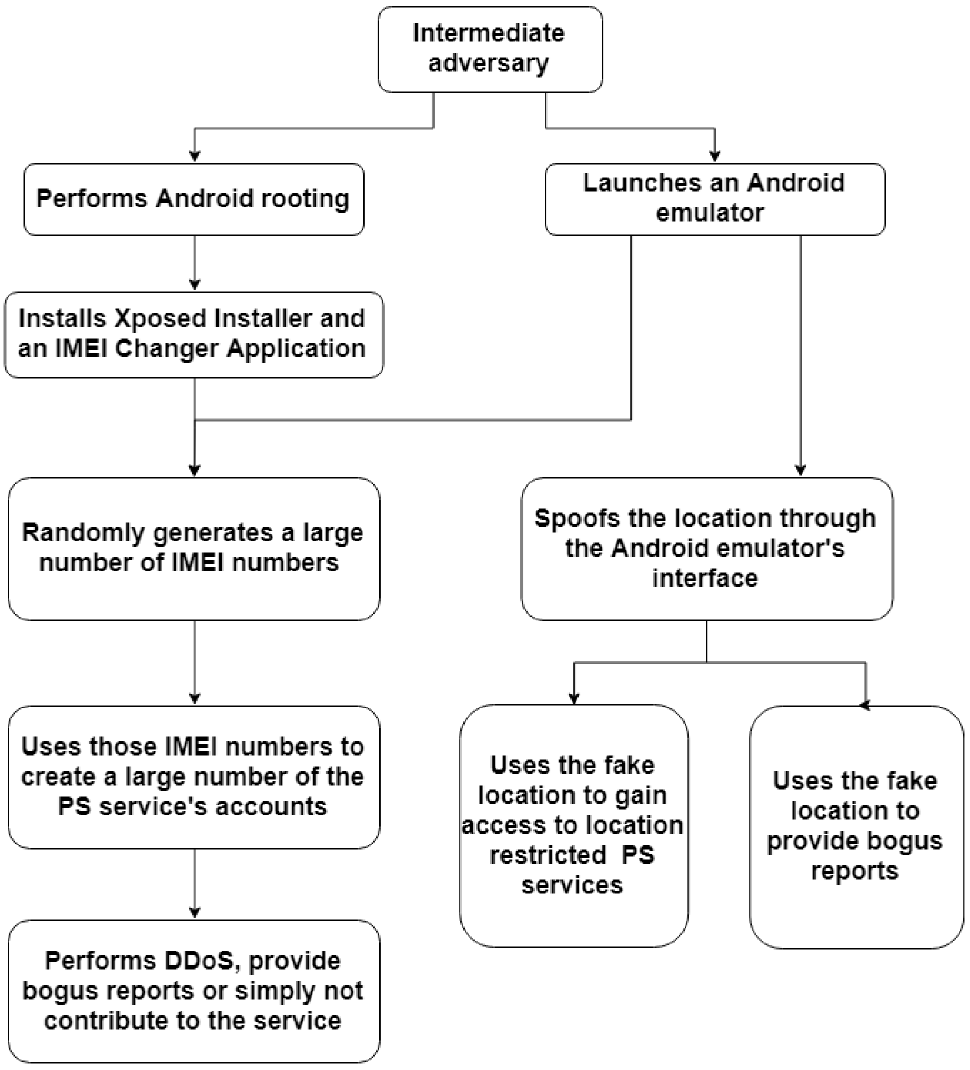

Intermediate adversary
Performs Android rooting
Launches an Android emulator
Installs Xposed Installer and an IMEI Changer Application
Randomly generates a large number of IMEI numbers
Spoofs the location through the Android emulator's interface
Uses those IMEI numbers to create a large number of the PS service's accounts
Uses the fake location to gain access to location restricted PS services
Uses the fake location to provide bogus reports
Performs DDoS, provide bogus reports or simply not contribute to the service

Figure 4.2: Attack procedures by intermediate adversary

to degrade the accuracy of the local information in the participatory sensing service.

2. The adversary uses a rooted device or an emulator to randomly spoof the IMEI number to gain access to a participatory sensing service without using any evasion technique. The adversary can create a number of fake accounts to perform a Distributed Denial-of-Service (DDoS) attack, provide a mass of fake information or simply not contribute to the services.

4.2.3 The result of the attack with the rooted smartphone on the IMEI number

By using the rooted smartphone, the adversary can now install Xposed Installer and IMEI Changer Application which needs a superuser privilege to randomly change IMEI number via interface of the application by following this procedure [46]. At the time that the hardened registration process checked the signing-up HTC Desire S smartphone, it found that it is a rooted smartphone. Malicious applications such as superSU and Xposed Installer for rooting, and IMEI Changer for IMEI spoofing are detected from detecting malicious installed package information with getPackageInfo. The SU path that is acquired from the rooting process is detected by searching the SU path in the suspected directories. The checking directory permissions technique also detected the writability permission in the suspected directories that only superuser privileges can gain access to. However, the technique of rooting that is used in this test can evade the checking the system properties technique by itself. Therefore, the rooted smartphone without any evasion techniques can be detected by rooting detector in the hardened registration process before the adversary attempts to spoof the IMEI number.

4.2.4 The result of the attack with the emulators on the location and IMEI number

By using the emulators, the adversary can easily choose any IMEI number and any location via interface of the applications to gain access to the participatory sensing service. All emulators can be detected via Bluetooth detecting technique. The Bluetooth detecting technique is an outstanding feature which can distinguish the difference between emulators and real smartphones since all tested emulators cannot emulate Bluetooth on it. Checking Android operating system properties is also useful for detecting emulation behavior. In other words, the emulator detector found that all emulators except the Nox emulator has the values of ro.kernel.qemu = 1 , ro.secure = 0 , ro.debuggable = 1 and service.adb.root = 1. Other methods that can support the detection are checking the voice mail number and line1 number. The Nox emulator and Android Studio Emulator returned +15552175049 for the voice mail number. The Android Studio Emulator returned 15555215554 for the line1 number. Therefore, the emulators without any evasion techniques can be detected by the rooting detector in the hardened registration process before the adversary makes any changes on the location and IMEI number.

4.3 Attacking by the advanced adversary

4.3.1 Adversarial tools

Now the adversary exploits the use of a rooted device to occupy the participatory sensing service with some rooting evasion techniques. This test changes a HTC Desire S from a legitimate smartphone to a rooted smartphone by using this procedure [47] with Hide my Root [48] and Hide Rooting Lite [49] to cloak the superuser privilege.

4.3.2 Attacking methods

The adversary uses a rooted device or an emulator to randomly spoof the IMEI number to gain access to a participatory sensing service. In addition, this test installs Hide my Root and Hide Rooting Lite on the rooted smartphone to cloak the superuser privilege and simply start to run each of these rooting evasion applications independently. The adversary can create a number of fake accounts to perform a Distributed Denial-of-Service (DDoS) attack, provide a mass of fake information or simply not contribute to the services.

4.3.3 The result of the attack with the rooted smartphone on the IMEI number

The adversary installs Xposed Installer and IMEI Changer Application to randomly change the IMEI number via interface of the application and start Hide my Root or Hide Rooting Lite. By using either Hide my Root or Hide

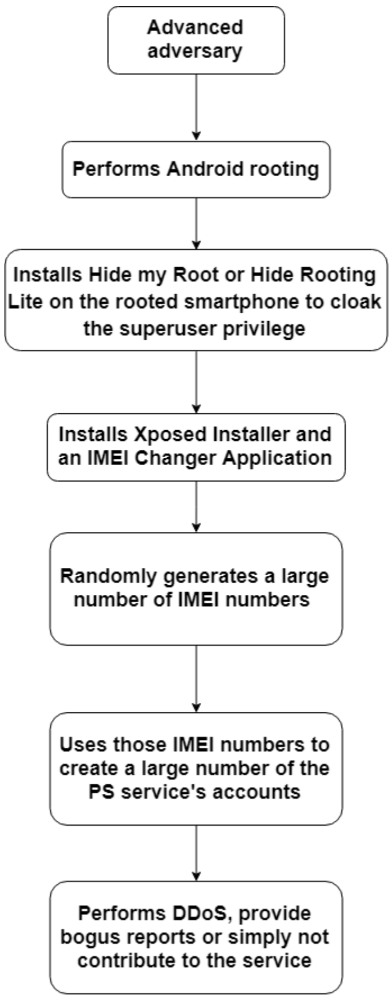

Advanced adversary
Performs Android rooting
Installs Hide my Root or Hide Rooting Lite on the rooted smartphone to cloak the superuser privilege
Installs Xposed Installer and an IMEI Changer Application
Randomly generates a large number of IMEI numbers
Uses those IMEI numbers to create a large number of the PS service's accounts
Performs DDoS, provide bogus reports or simply not contribute to the service

Figure 4.3: Attack procedures by advanced adversary

Rooting Lite, the root detector in the hardened registration process can detect the superuser privilege via detecting malicious installed package information with getPackageInfo. On the other hand, Hide Rooting Lite can cloak the superuser privilege from the checking directory permissions technique, and both Hide my Root and Hide Rooting Lite can cloak the superuser privilege from the searching the SU path technique. Therefore, the rooted smartphone with these evasion techniques can be detected by rooting detector in the hardened registration process before the adversary attempts to spoof the IMEI number. However, if the list of malicious installed packages does not cover newer malicious applications, the rooted smartphone may not be detected by the rooting detector.

In the case that the adversary uses other effective rooting evasion techniques and passes through the rooting detector, IMEI spoofing can still be detected by the

IMEI inspector if the adversary does not have any knowledge about the IMEI number system and change the IMEI number randomly.

4.4 Attacking by the professional adversary

4.4.1 Adversarial tools

It is assumed that the adversary exploits the use of a rooted device with some rooting evasion techniques that can avoid the rooting detection or with a customized emulator that can emulate Bluetooth to avoid the emulator detection. In addition, the adversary also has knowledge about the IMEI number system.

4.4.2 Attacking methods

The adversary uses a rooted device or an emulator to spoof the IMEI number to gain access to a participatory sensing service with knowledge about the IMEI number system. The adversary succeeds to avoid the rooting detection or the emulator detection and changes an IMEI number that matches with the rooted smartphone or the virtual smartphone model information via the interface of the IMEI changer application on the rooted smartphone or the interface of the customized emulator.

4.4.3 The result of the attack with the rooted smartphone and the customized emulator on the IMEI number

Since the adversary can pass through the rooting detection, the emulator detection and IMEI inspector, the adversary would get access to a participatory sensing service to degrade the participatory sensing service with the rooted smartphone. However, the customized emulator still can be detected by botnet detector if the customized emulator cannot emulate some natural human movements such as true shaking, multi-touching and tilting which can be distinguished by

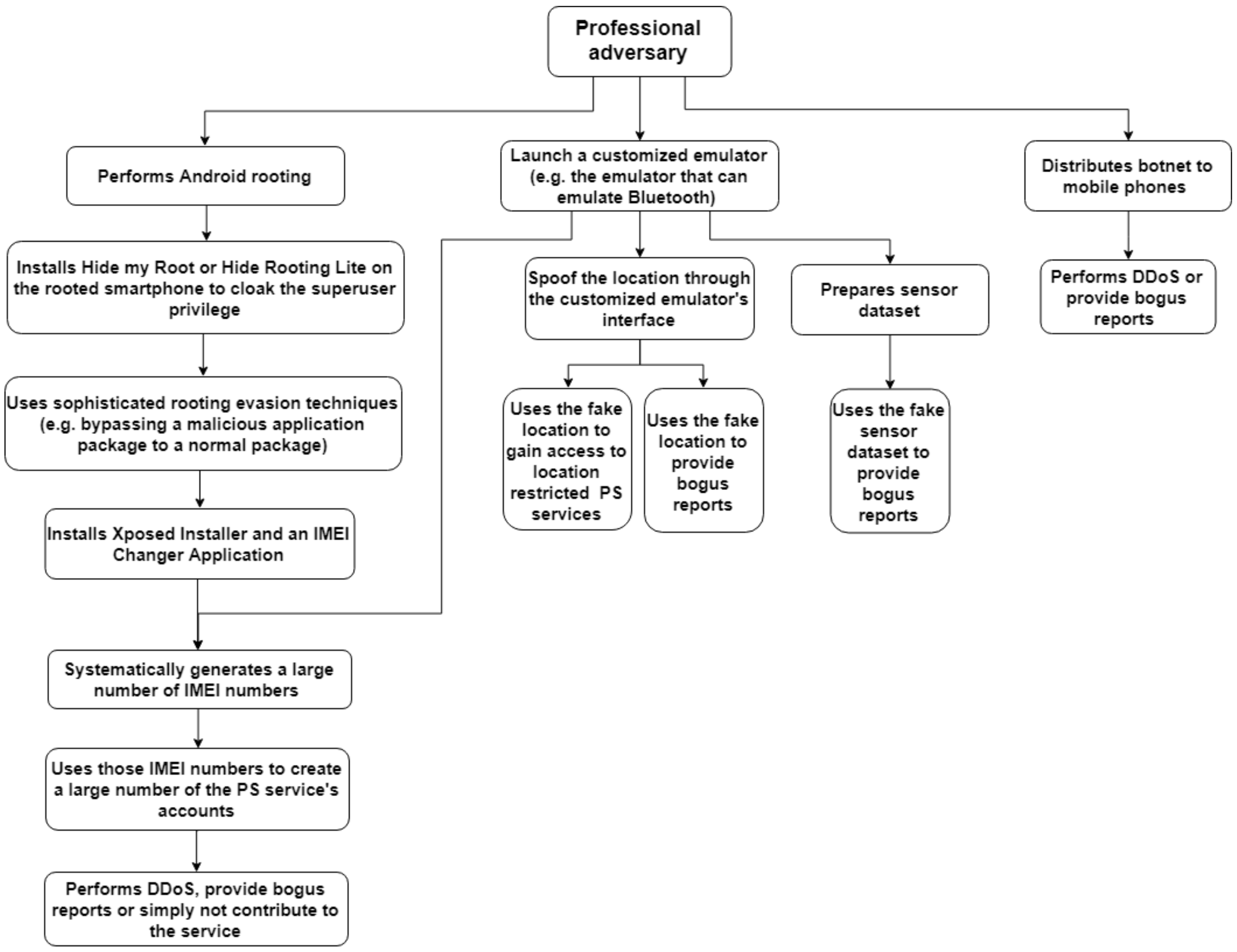

Professional adversary
Performs Android rooting
Installs Hide my Root or Hide Rooting Lite on the rooted smartphone to cloak the superuser privilege
Uses sophisticated rooting evasion techniques (e.g. bypassing a malicious application package to a normal package)
Installs Xposed Installer and an IMEI Changer Application
Systematically generates a large number of IMEI numbers
Uses those IMEI numbers to create a large number of the PS service's accounts
Performs DDoS, provide bogus reports or simply not contribute to the service
Launch a customized emulator (e.g. the emulator that can emulate Bluetooth)
Spoof the location through the customized emulator's interface
Uses the fake location to gain access to location restricted PS services
Uses the fake location to provide bogus reports
Prepares sensor dataset
Uses the fake sensor dataset to provide bogus reports
Distributes botnet to mobile phones
Performs DDoS or provide bogus reports

Figure 4.4: Attack procedures by professional adversary

MoCAPTCHA or cannot solve the gyroscope mini game which is actually hard to solve by a virtual device (a personal computer).

4.5 The result of using smartphones with imperfect sensor condition

This case may not be a malicious intention. However, in order to receive good quality information for the participatory service, the smartphone that has the right to gain access must have at least a camera, location sensors and motion sensors with perfect conditions.

Since all of the sample smartphones have those basic sensors with perfect conditions, we created a scenario where a barometer would also need to be detected by assuming that the service needs to measure atmospheric pressure.

All of the sample smartphones are rejected by the sensor inspector because none of it has a barometer.

This is only the functional test of the sensor inspector to prove that it is working. In reality, we can only inspect a global positioning system, a network-based geolocation system, a camera, an accelerometer and a magnetometer for basic information.

4.6 The result of the attack by a botnet

When the adversary deploys a botnet via a number of smartphones to gain access to the participatory sensing service and tries to perform a DDoS attack on the participatory sensing service, the botnet will be detected by the botnet detector. MoCAPTCHA tasks need human motions to solve which the botnet cannot perform. In the case that the smartphone is infected by an intelligent botnet which can solve the MoCAPTCHA tasks by, somehow, achieving multi-touch tasks and emulating human shaking and human tilting, it is still difficult for this smart botnet to solve the gyroscope mini game which requires full control on the smartphone to solve ad hoc tasks with limited attempts.

Each level of the adversaries has different capabilities to penetrate the hardened registration process. Table 9 shows the effectiveness of each detection and inspection technique in our checks and controls according to each type of attack.

Countermeasures		Beginner adversary	Intermediate adversary			Advanced adversary		Professional adversary					Using smartphones with imperfect sensor condition
Checks and controls	Techniques	GNSS location spoofer apps	Using rooted devices	Using Android emulators	Randomly change IMEI number	Using rooting evasion apps	Randomly change IMEI number	Using a customized emulator for Bluetooth emulation	Using botnet	Change IMEI number with knowledge	Using rooting evasion apps and bypassing malicious packages to normal packages	Prepare sensor data	
Rooting detector	Searching for suspected packages		O			O					X		
	Checking directory permissions		O			Δ					Δ		
	Checking system properties		Δ			X					X		
	Searching for the SU path		O			X					X		
Emulator detector	Detecting emulation through Bluetooth			O				X					
	Detecting emulation through Android API			Δ				Δ					
	Checking system properties			Δ				Δ					
Sensor inspector	Checking system features			O				X				X	O
	Searching for ”Goldfish”			Δ				Δ				X	
IMEI inspector	Proving phone model with TAC database				O		O			X			
	Checking duplicated IMEI number				Δ		Δ			X			
Location inspector	Checking location manipulation	O											
	Proving phone location with a target area boundary	O											
Botnet detector	MoCAPTCHA			Δ				Δ	O			O	O
	Gyroscope mini game			Δ				Δ	O			O	O

Table 9: The result of the attacks on the hardened registration process

** O = Can be addressed, Δ = Might be addressed and X = Cannot be addressed.

4.7 Time consumption on the smartphone inspection activity

The rooting detector, emulator detector, sensor inspector, IMEI inspector and location inspector have been combined into an activity. The activity has been tested on three different Android phones which are Samsung GT-I9082L, Samsung A5 and HTC Desire S. Each device ran the activity for 10 times to determine the average running time, maximum running time and minimum running where the results are illustrated in table 10.

Devices	Maximum running time	Minimum running	Average running time
Samsung GT-I9082L	50.86 seconds	40.46 seconds	43.95 seconds
Samsung A5	45.56 seconds	40.59 seconds	41.33 seconds
HTC Desire S	46.14 seconds	42.79 seconds	44.01 seconds

Table 10: The running time on the phone inspection activity.

4.8 User reviews on the botnet detector

To get the feedback from users on the botnet detector, we analyse mobile CAPTCHA heuristics with factors that were offered in [50] that are:

1. “User control and freedom – Input mechanisms required to answer the CAPTCHA should not lead users to make mistakes.”
2. “Learnability – The scheme should provide, and not require more than, brief instructions. The CAPTCHA scheme should be intuitive, without excessive cognitive load.”
3. “Efficiency of use – The scheme should be quick and efficient to use. The scheme should avoid controls or non-CAPTCHA related artifacts that may lead to errors or inefficiencies, and minimize unnecessary complexity.”
4. “Input mechanisms – The scheme should support input mechanisms allowing easy completion of the challenge on different devices.”
5. “Solvability - CAPTCHA challenges should be simple for users to solve while preventing automated computer solutions.”

6. “User perception – CAPTCHAs should be pleasant to solve and should cause no discomfort to users.”
7. “Consistency with user’s localization and environment – The scheme should be universal and should be suitable for the range of situations and environments in which users may access the scheme. Language and culture should not impose additional barriers to solving challenges.”

However, to present the feedback in details, we are going to show qualitative feedbacks. The MoCAPTCHA and the gyroscope mini game modules were tested by six testers. The testers solved these modules two times with Samsung GT-I9082L for the first time and with Samsung A5 for the second time. No hints were given to the tester during the test. After the tester had to answer the questionnaire for each module which relates to the mentioned factors.

4.8.1 MoCAPTCHA

The feedbacks from the users on the MoCAPTCHA are shown in table 11.

Mobile CAPTCHA heuristics’ factors	Positive feedback	Negative feedback
User control and freedom	- The interface of MoCAPTCHA is quite clear and interacts with the testers smoothly. - There is no concern on touching screen to choose circles.	- The majority of the testers coincides that solving the MoCAPTCHA requires some attention which is difficult to solve while they are walking. - Three tester need to place the smartphone on a standstill place to solve the MoCAPTCHA. - There is a minor concern on two tester who confuse about recognizing the circles and the ovals.
Learnability	- With brief instructions on the MoCAPTCHA, it is easy to solve the scheme.	
Efficiency of use	- There is no concern on time consuming. All tester said that they used a reasonable amount of time to solve the MoCAPTCHA. - The record of the maximum, average and minimum time that the testers used to solve the MoCAPTCHA on the first time is 74.41, 45.83 and 28.41 seconds respectively.	
Input mechanisms	- The majority has no concern on solving the MoCAPTCHA on the different smartphone.	- There is a minor concern on two tester who said that it might be a bit more difficult to solve. (e.g. multi-touching and shaking tasks) if the testers had to solve it on bigger smartphones or tablets due to the size of the devices.
Solvability	- There is no distractor that makes the MoCAPTCHA too difficult to solve. - There is no confusion on what the testers have to do.	
User perception	- All testers do not feel uncomfortable to solve the MoCAPTCHA. - The majority of the testers said that the MoCAPTCHA is interesting to solve.	
Consistency with the user’s localization and environment		- If the users do not understand the language of the instructions , it might be difficult to figure out how to solve it.

Table 11: The MoCAPTCHA heuristics analysis.

4.8.2 Gyroscope mini game

The feedbacks from the users on the MoCAPTCHA are shown in table 12.

Mobile CAPTCHA heuristics' factors	Positive feedback	Negative feedback
User control and freedom	- The interface of the gyroscope mini game is clear for the user to interact with.	- The majority of the testers coincides that solving the gyroscope mini game requires some attention which is difficult to solve while they are walking. - The majority of the testers coincides that the ball in the game was a bit difficult to control and responded to extreme movements
Learnability	- Even though there were no instructions on the gyroscope mini game, it is easy to solve the scheme.	
Efficiency of use	- There is no concern on time consumption. All tester said that they used a reasonable amount of time to solve the gyroscope mini game. - The record of the maximum, average and minimum time that the testers used to solve the gyroscope mini game at the first time is 26.71, 17.89 and 10.46 seconds respectively.	
Input mechanisms	- The majority of the testers has no concern on solving the gyroscope mini game on the different smartphone.	
Solvability	- There is no distractor that makes the gyroscope mini game too difficult to solve. - There is no confusion on what the testers have to do.	- The majority the testers coincides that solving some random tasks in the gyroscope mini game is a bit difficult. Some of game sets made them fail the test.
User perception	- All testers do not feel uncomfortable to solve the gyroscope mini game. - The majority of the testers said that the gyroscope mini game is interesting and fun to solve.	- The majority of the testers coincides the difficulty of the gyroscope mini game also depends on whether the users have played this kind of the game before or not.
Consistency with user's localization and environment	- The majority of the testers said that there is no need of using any instruction for the gyroscope mini game. It is learnable intuitively.	

Table 12: The gyroscope mini game heuristics analysis.

5 Conclusion & Discussion

The openness of the registration process can become a vulnerability for participatory sensing services because the adversary could overwhelm the participatory sensing services to degrade the quality of the shared/contributed data. The hardened registration process offers a process with defensive measures towards rejecting spurious registrations to protect participatory sensing services from abuse.

The hardened registration process makes it significantly harder for such adversarial behavior to be effective while the process does not force the users to provide their sensitive information. Checks and controls are complementary and reinforce one another to reduce the choices of using adversarial tools with various detection and inspection techniques. This combination can stop the adversary who exhibits suspicious behavior by owning illegitimate devices even before the adversary starts to perform a malicious action.

From the result of the security and performance analysis, the hardened registration process can protect the participatory sensing service by blocking the malicious tools that were used by the beginner adversary to the advance adversary. Moreover, even though the adversaries can evade the adversarial tools from the detectors with sophisticated techniques, the malicious actions can be blocked as long as the adversaries do not have knowledge about the IMEI number system or location mechanism. However, for the professional adversary, knowing the IMEI number system can be the weakness of the hardened registration process. From the heuristics analysis, the MoCAPTCHA and the gyroscope mini game in the botnet detector work functionally and interact well with the smartphone users.

6 Future work

6.1 The development of checks and controls in the hardened registration process

There is a gap of development on each check and control. For the rooting inspector, the development between the detection techniques and the rooting evasion techniques are always competitive. Thus, upgrading the rooting inspector is needed in the future. But there is one possible solution. The misbehavior checking on rooting can be adopted at kernel level which is the level that the smartphone hardware and the smartphone software communicate with each other.

For the emulator inspector, we can develop detection techniques on the emulating behavior that does not only rely on the Bluetooth detection. One example is checking emulating behavior through hardware performances such as checking graphic performance.

Since knowing the IMEI number system can be the weakness of the hardened registration process, the development of a secured protocol for approving the IMEI number is needed. The IMEI number of the device can be encrypted with a secret key that is restrictively distributed by a participatory sensing campaign, sent though other entities in the participatory sensing infrastructure and decrypted and checked by the campaign.

To strengthen the security on the location mechanism, the location inspector can be also supported by other location references such as local landmarks and magnetic field.

6.2 The development of post-registration process

Every participatory sensing service needs some contribution from the participants. Monitoring the consistency of reporting or querying information for the participants can help participatory sensing services ensure accurate information. Therefore, the development of the reputation system for the participatory sensing system is interesting.

Misbehaviors from false information reports, spamming requests, the infection of botnets and repetitive enormous reports in the post-registration level can happen. Some checks and controls in the hardened registration process can be applied with other new techniques that suit the post-registration process in collecting accurate information.

As mentioned before, the limitation of the hardened registration process is that this work cannot protect the participatory sensing systems from the adversaries who use real devices. They may perform misbehaviors which are mentioned above or recruit a large number of volunteers to get access to the service and to perform DDoS later. The post-registration process can be a solution for this problem as well. For example, tracking the location and timing of new entry devices can give information on patterns relating to the registration behavior of these new participants. Moreover, learning and capturing the pattern of participants' actions which lead to malicious and erroneous behavior can stop the adversaries before they perform attacks.

References

[1] A. Vasilateanu et al. "Environment crowd-sensing for asthma management". In: *2015 E-Health and Bioengineering Conference (EHB)* (2015).

[2] Alina Sîrbu et al. "Participatory Patterns in an International Air Quality Monitoring Initiative". In: *PLOS ONE* 10.8 (2015), e0136763. DOI: 10.1371/journal.pone.0136763.

[3] Zheng Xu et al. "Participatory sensing-based semantic and spatial analysis of urban emergency events using mobile social media". In: *EURASIP Journal on Wireless Communications and Networking* 2016.1 (2016). DOI: 10.1186/s13638-016-0553-0.

[4] Shin'ichi Konomi et al. "User Participatory Sensing for Disaster Detection and Mitigation in Urban Environments". In: *Distributed, Ambient and Pervasive Interactions* (2016), pp. 459–469. DOI: 10.1007/978-3-319-39862-4_42.

[5] Pengfei Zhou, Zhiyuan Chen, and Mo Li. "Smart traffic monitoring with participatory sensing". In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys '13* (2013). DOI: 10.1145/2517351.2517379.

[6] Zhidan Liu et al. "A Participatory Urban Traffic Monitoring System: The Power of Bus Riders". In: *IEEE Transactions on Intelligent Transportation Systems* 18.10 (2017), pp. 2851–2864. DOI: 10.1109/tits.2017.2650215.

[7] *International Mobile Equipment Identity*. https://en.wikipedia.org/wiki/International_Mobile_Equipment_Identity. 2018.

[8] Lawrence C. Miller. *DDoS For Dummies*. Hoboken, New Jersey: John Wiley & Sons, Inc, 2012, pp. 3–14.

[9] J. Hidalgo and G. Alvarez. "CAPTCHAs". In: *Advances in Computers* (2011), pp. 109–181.

[10] D. Estrin et al. “Participatory sensing: applications and architecture [Internet Predictions]”. In: *IEEE Internet Computing* 14.1 (Jan. 2010), pp. 12–42. ISSN: 1089-7801. DOI: `10.1109/MIC.2010.12`.

[11] Jeffrey Goldman et al. *Participatory sensing: A citizen-powered approach to illuminating the patterns that shape our world*. Washington, D.C.: Woodrow Wilson International Center for Scholars, 2009.

[12] E. De Cristofaro and C. Soriente. “Participatory privacy: Enabling privacy in participatory sensing”. In: *IEEE Network* 27.1 (2013), pp. 32–36. DOI: `10.1109/mnet.2013.6423189`.

[13] *Number of smartphones sold to end users worldwide from 2007 to 2017 (in million units)*. https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. 2018.

[14] *3.5 million photos shared every minute in 2016*. https://www2.deloitte.com/uk/en/pages/press-releases/articles/3-point-5-million-photos-shared-every-minute.html. 2018.

[15] S. Sitharama Iyengar et al. “Wireless Sensor Networks”. eng. In: *Fundamentals of Sensor Network Programming*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Sept. 2010, pp. 15–26. ISBN: 9780470876145.

[16] Katherine A. Johnson. “Real Life Science with Dandelions and Project BudBurst”. In: *Journal of Microbiology & Biology Education* 17.1 (2016), pp. 115–116. DOI: `10.1128/jmbe.v17i1.1064`.

[17] S. Gisdakis et al. “SHIELD”. In: *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec ’15. 2015.

[18] *Conditional random field*. https://en.wikipedia.org/wiki/Conditional_random_field. 2018.

[19] Qiang Ma et al. “PLP: Protecting Location Privacy Against Correlation Analyze Attack in Crowdsensing”. In: *Mobile Computing, IEEE Transactions on* 16.9 (2017), pp. 2588–2598.

[20] S. Gisdakis et al. “SPPEAR”. In: *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*. WiSec ’14. 2014.

[21] *what is captcha (completely automated public turing test to tell computers and humans apart)?* https://searchsecurity.techtarget.com/definition/CAPTCHA. 2018.

[22] *ReCAPTCHA*. https://en.wikipedia.org/wiki/ReCAPTCHA. 2018.

[23] *NuCAPTCHA*. https://en.wikipedia.org/wiki/NuCaptcha. 2018.

[24] S. Hachem, A. Pathak, and V. Issarny. "Probabilistic registration for large-scale mobile participatory sensing". In: IEEE, 2013, pp. 132–140. ISBN: 9781467345736.

[25] Teemu Leppänen et al. "Mobile crowdsensing with mobile agents". In: *Autonomous Agents and Multi-Agent Systems* 31.1 (2017), pp. 1–35. ISSN: 1387-2532.

[26] Sasank Reddy, Deborah Estrin, and Mani Srivastava. "Recruitment Framework for Participatory Sensing Data Collections". In: *Proceedings of the 8th International Conference on Pervasive Computing*. Pervasive'10. Helsinki, Finland: Springer-Verlag, 2010, pp. 138–155. ISBN: 3-642-12653-7, 978-3-642-12653-6. DOI: `10.1007/978-3-642-12654-3_9`. URL: `http://dx.doi.org/10.1007/978-3-642-12654-3_9`.

[27] X. Ignatius Selvarani et al. "Secure voting system through SMS and using smart phone application". In: vol. 2017. IEEE, Feb. 2017, pp. 1–3. ISBN: 9781509033782.

[28] *International mobile subscriber identity*. https://en.wikipedia.org/wiki /International_mobile_subscriber_identity. 2018.

[29] *One-time password*. https://en.wikipedia.org/wiki/One-time_password. 2018.

[30] "Introduction". In: *GNSS — Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. Vienna: Springer Vienna, 2008, pp. 1–12. ISBN: 978-3-211-73017-1. DOI: `10.1007/978-3-211-73017-1_1`.

[31] San-Tsai Sun, Andrea Cuadros, and Konstantin Beznosov. "Android Rooting: Methods, Detection, and Evasion". In: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and*

Mobile Devices. SPSM '15. Denver, Colorado, USA: ACM, 2015, pp. 3–14. ISBN: 978-1-4503-3819-6. DOI: 10.1145/2808117.2808126.

[32] Gang Wang et al. "Ghost Riders: Sybil Attacks on Crowdsourced Mobile Mapping Services". In: *Networking, IEEE/ACM Transactions on* 26.3 (June 2018), pp. 1123–1136. ISSN: 1063-6692.

[33] Long Nguyen-Vu et al. "Android Rooting: An Arms Race between Evasion and Detection". In: *Security and Communication Networks* 2017 (2017). ISSN: 1939-0114.

[34] *Type Allocation Code*. https://en.wikipedia.org/wiki/Type_Allocation_Code. 2018.

[35] *Accessing Data with MongoDB: 2018*. https://spring.io/guides/gs/accessing-data-mongodb. 2018.

[36] Timothy Vidas and Nicolas Christin. "Evading Android Runtime Analysis via Sandbox Detection". In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '14. Kyoto, Japan: ACM, 2014, pp. 447–458. ISBN: 978-1-4503-2800-5. DOI: 10.1145/2590296.2590325.

[37] J. Shokouh. "Detecting GNSS Attacks on Smartphones". MA thesis. KTH, School of Electrical Engineering (EES), Communication Networks, 2013.

[38] *Fake GPS location*. https://play.google.com/store/apps/details?id=com.lexa.fakegps&hl=en. 2018.

[39] *Fake GPS GO Location Spoofer*. https://play.google.com/store/apps /details?id=com.incorporateapps.fakegps.fre&hl=en. 2018.

[40] *XPOSED IMEI Changer*. https://play.google.com/store/apps /details?id=com.vivek.imeichanger&hl=en. 2018.

[41] *MTK Engineering Mode*. https://play.google.com/store/apps /details?id=com.themonsterit.EngineerStarter&hl=en. 2018.

[42] *Bluestacks - The Best Android Emulator on PC as Rated by You*. https://www.bluestacks.com/bluestacksgaming-platform-bgp-android-emulator.html. 2018.

[43] *Run apps on the Android Emulator*. https://developer.android.com/studio/run/emulator. 2018.

[44] *MEmu App Player*. http://www.memuplay.com/. 2018.

[45] *Nox App Player*. https://www.bignox.com/. 2018.

[46] Jitender Singh. *How to Change Android IMEI Number (Root - Without Root*. https://www.viralhax.com/change-imei-number-android/. 2018.

[47] *How to root, install ext4 recovery and custom rom on hboot 2.00.0002*. https://forum.xda-developers.com/showthread.php?t=1525100. 2012.

[48] *Hide my Root*. https://apkpure.com/hide-my-root/com.amphoras.hidemyroot. 2018.

[49] *Hide Rooting Lite*. https://play.google.com/store/apps /details?id=com.formyhm.hideroot&hl=en. 2018.

[50] Gerardo Reynaga, Sonia Chiasson, and Paul van Oorschot. “Heuristics for the evaluation of captchas on smartphones”. In: *Proceedings of the 2015 British HCI Conference*. British HCI ’15. ACM, July 2015, pp. 126–135. ISBN: 9781450336437.

7 Appendices

Appendix - Contents

A Rooting detector's pseudo code

```
public static final String[] RootPackages = {
        /**
        *** The list of known rooting application packages.
        **/
};
public static final String[] suPaths ={
        /**
        *** The list of suspected directories that "su" might be
            located.
        **/
};
public static final String[] Unwritable = {
        /**
        *** The list of suspected directories that should not be
            writable.
        **/
};
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    if (RootApps()||DangerousProps()||Writablility()||Binary("su")){
        info = "This device is rooted!";
        // Reject the participant.
    }else {
        info = "This device is not rooted!";
    }
}
```

```
public boolean RootApps() {
    ArrayList<String> packages = new ArrayList<>();
    packages.addAll(Arrays.asList(RootPackages)); // listing rooting
        application packages.
    boolean result = false; // Set "false" as default.
    /**
    *** Get application packages by getPackageInfo(); method and
        compare with the RootPackages list.
    *** Return result = ture; if the rooting application is detected.
   **/
}
```

```
private String[] ReadProp(){
  /**
  *** Implement Runtime.getRuntime().exec("getprop").getInputStream();
      to get the list of properties in the input stream.
   **/
    return InputSteam;
}
public boolean DangerousProps(){
    final Map<String, String> dangerousProps = new HashMap<>();
    dangerousProps.put("ro.debuggable","1");
    dangerousProps.put("ro.secure","0");
    dangerousProps.put("ro.kernel.qemu","1");
    dangerousProps.put("service.adb.root","1");
    dangerousProps.put("persist.service.adb.enable","1");
    boolean result = false; // Set "false" as default.
    /**
     *** Check the input stream from ReadProp() method and compare
         with dangerousProps.
     *** Return result = ture; if the dangerousProps was found.
   **/
}
```

```
private String[] Readmount() {
  /**
  *** Implement Runtime.getRuntime().exec("mount").getInputStream();
      to get the mount options of directories in the input stream.
   **/
    return InputSteam;
}
public boolean Writablility() {
    boolean result = false; // Set "false" as default.
  /**
     *** Check the input stream from Readmount() method and compare
         the mount options of Unwritable directories list against "rw".
     *** Return result = ture; if "rw" was found in the mount options
         from any Unwritable directories.
   **/
}
```

```
public boolean Binary(String filename) {
    String[] pathsArray = suPaths;
    boolean result = false; // Set "false" as default.
    for (String path : pathsArray) {
        File f = new File(path, filename);
        boolean fileExists = f.exists();
        if (fileExists) {
            result = true;
        }
    }return result;
}
```

B Emulator detector’s pseudo code

```
private static final String[] PHONE_NUMBERS = {
        "15555215554", "15555215556", "15555215558", "15555215560",
            "15555215562", "15555215564","15555215566", "15555215568",
            "15555215570", "15555215572", "15555215574",
            "15555215576","15555215578", "15555215580", "15555215582",
            "15555215584"
};

private static final String[] DEVICE_IDS = {
        "000000000000000","e21833235b6eef10","012345678912345"
};
private static final String[] VM_NUMBER = {
        "+15552175049"
};
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    BluetoothAdapter mBluetoothAdapter =
        BluetoothAdapter.getDefaultAdapter();
    if (mBluetoothAdapter == null){
        bluetooth = "null";
    }else {
        bluetooth = mBluetoothAdapter.getName();
    }

    if (DangerousProps()||checkID()||checkPhoneNumber()||
    checkVMNumber()||mBluetoothAdapter == null){
        Emulator = "This is an emulator!";
         // Reject the participant.
    }else {
        Emulator = "This is not an emulator!";
    }
```

```java
public boolean checkPhoneNumber() {
    TelephonyManager telephonyManager1 = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);
    String phoneNumber = telephonyManager1.getLine1Number();
    for (String number1 : PHONE_NUMBERS) {
        if (number1.equalsIgnoreCase(phoneNumber)) {
            return true;
        }
    }
    return false;
}
```

```java
public boolean checkVMNumber() {
    TelephonyManager telephonyManager2 = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);
    String VMNumber = telephonyManager2.getVoiceMailNumber();
    for (String number2 : VM_NUMBER) {
        if (number2.equalsIgnoreCase(VMNumber)) {
            return true;
        }
    }
    return false;
}
```

```
public boolean checkID() {
    TelephonyManager telephonyManager3 = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);
    String Device_id = telephonyManager3.getDeviceId();
    for (String number3 : DEVICE_IDS) {
        if (number3.equalsIgnoreCase(Device_id)) {
            return true;
        }
    }
    return false;
}
```

```
private String[] ReadProp(){
  /**
  *** Implement Runtime.getRuntime().exec("getprop").getInputStream();
      to get the list of properties in the input stream.
   **/
    return InputSteam;
}
public boolean DangerousProps(){
    final Map<String, String> dangerousProps = new HashMap<>();
    dangerousProps.put("ro.debuggable","1");
    dangerousProps.put("ro.secure","0");
    dangerousProps.put("ro.kernel.qemu","1");
    dangerousProps.put("service.adb.root","1");
    dangerousProps.put("persist.service.adb.enable","1");
    boolean result = false; // Set "false" as default.
    /**
     *** Check the input stream from ReadProp() method and compare
         with dangerousProps.
     *** Return result = ture; if the dangerousProps was found.
   **/
}
```

C Sensor inspector's pseudo code

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    PackageManager pm = getPackageManager();
    final boolean deviceHasCameraFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_CAMERA);
    final boolean deviceHasGPSFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_LOCATION_GPS);
    final boolean deviceHasNetFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_LOCATION_NETWORK);
    final boolean deviceHasAccFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_ACCELEROMETER);
    final boolean deviceHasCompassFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS);
    final boolean deviceHasBaroFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_BAROMETER);
    final boolean deviceHasBtFlag =
        pm.hasSystemFeature(PackageManager.FEATURE_BLUETOOTH);

    if((deviceHasCameraFlag)&&(deviceHasGPSFlag)&&(deviceHasNetFlag)
    &&(deviceHasAccFlag)&&(deviceHasCompassFlag)&&(deviceHasBaroFlag)
    &&(deviceHasBtFlag)){
        sensor = "The sensors are fine!";
    }else {
        sensor = "Imperfect!";
        // Reject the participant.
    }
}
```

D IMEI inspector's pseudo code

```
@Override
protected void onCreate(Bundle savedInstanceState) {

        TelephonyManager manager = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);
        String IMEINumber = manager.getDeviceId();//Get IMEI number
            from the smartphone.

        if (!TextUtils.isEmpty(IMEINumber)){
            TACnum = IMEINumber.substring(0,8);
            SERIALnum = IMEINumber.substring(8);
            //Get TAC and serial number from the IMEI number.
        }

        PhoneBrand = Build.BRAND;
        PhoneModel = Build.MODEL;
        //Get Brand and Model from the smartphone.

        new HttpRequestTask1().execute(); //Access TAC database
        new HttpRequestTask2().execute(); //Access registered user
            database
    }
}
```

```
private class HttpRequestTask1 extends AsyncTask<Void , Void, String>{
  /**
     *** Send a TAC as a query to the TAC database in AWS through a
         Tomcat web service.
       *** Get a brand and a model that relate to the TAC.
       *** Compare the brand and the model from the TAC database with
           Build.BRAND and Build.MODE
   **/
        if(TACbrand.toLowerCase().contains(PhoneBrand.toLowerCase())
        &&TACmodel.toLowerCase().contains(PhoneModel.toLowerCase())){
            TAC = "This is a real IMEI.";
        }else{
            TAC = "This is a spoofed IMEI.";
             // Reject the participant.
        }
    }
```

```
private class HttpRequestTask2 extends AsyncTask<Void, Void, String>{
  /**
     *** Send an IMEI number as a query to the registered user
         database in AWS through a Tomcat web service.
       *** Search for IMEI number from the registered user database.
   **/
        if(matching.toLowerCase().contains("true".toLowerCase())){
            user = "This user has been registered.";
            // Reject the participant.
        }else{
            user = "This user is a new user.";
        }
    }
```

E Location inspector's pseudo code

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
      0, 0, locationListener); //Listen to the phone's location

        TelephonyManager tel = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);
        GsmCellLocation cellLocation = (GsmCellLocation)
            tel.getCellLocation();
        int cellid = cellLocation.getCid(); //Get Cell-ID
        int celllac = cellLocation.getLac(); //Get a Location Area Code
        String subscriberID = tel.getSubscriberId(); //Get a Mobile
            Country Code and a Mobile Network Code

        if (!TextUtils.isEmpty(subscriberID)){
           //Split the Mobile Country Code and the Mobile Network Code
            MCCnum = subscriberID.substring(0,3);
            NETnum = subscriberID.substring(3,5);
            netNum = Integer.parseInt(NETnum);
        }

        CellID = String.valueOf(cellid);
        CellLAC = String.valueOf(celllac);
        MNCnum = String.valueOf(netNum);

        new HttpRequestTask1().execute(); //Access Cell-ID database
    }
```

```
private final LocationListener locationListener = new
    LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        double longitude = location.getLongitude();
        double latitude = location.getLatitude();
        //Get longitude and latitude from the global positioning
            system in the phone.
      }
    private class HttpRequestTask1 extends AsyncTask<Void , Void, String>
        {
        /**
         *** Send The Cell-ID, the LAC, the MCC and the MNC as a query to
             the Cell-ID database in AWS through a Tomcat web service.
           *** Get a latitude value and a longitude value that relate to
               the query.
       **/
            if((Math.abs(celllon-gpslon)<0.02)
            &&(Math.abs(celllat-gpslat)<0.01)){
            //Compare the location from Cell-ID database with the location
                from the global positioning system in the phone.
                location = "This is a true location";
                if((gpslon>17.715996)&&(gpslon<18.421821
                &&(gpslat>59.149776)&&(gpslat<59.509600)){
                //Target area
                    area = "This device locates in the target area.";
                }else{
                    area = "This device does not locate in the target area.";
                    // Reject the participant.
                }
            }else{
                location = "This is a spoofed location.";
                //Compare the location from Cell-ID database far from the
                    location from the global positioning system in the phone
                    further than 2 kilometers.
                // Reject the participant.
            }
        }
    }
}
```

F MoCAPTCHA's pseudo code

```
@Override
public void onCreate(Bundle savedInstanceState) {
/**
 *** Randomly generate the set of multi-touching in the first task.
 **/
mShaker = new ShakeListener(this);
mShaker.setOnShakeListener(new ShakeListener.OnShakeListener () {
//Listen to shaking event.
  public void onShake() {
     if((touchCanvas.isTouchAchieved()==true)&&(round==0)&&(no_bot==true)){
     //Check if all red circles has been touched in the first task and
         the tilt angle has changed.
        @Override
        public void onClick(DialogInterface dialogInterface, int
            which) {
         /**
         *** Randomly generate the set of multi-touching in the second
             task.
        **/
     }
     }else if((touchCanvas.isTouchAchieved()==true)
     &&(round==1)&&(no_bot==true)){
     //Check if all red circles has been touched in the second task
         and the tilt angle has changed.
     //Run a gyroscope mini game.
     } else {
     //Notice the user to touch the red circles.
     }
   });
  }
}
```

```
@Override
public void onSensorChanged(SensorEvent event){
 /**
 *** Check an azimuth value, a roll value and a pitch value to detect
     the tilt angle.
 **/
}
```

TRITA TRITA-EECS-EX-2018:608